NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

IMPLEMENTATION OF A TACTICAL MISSION PLANNER FOR COMMAND AND CONTROL OF COMPUTER GENERATED FORCES IN MODSAF

by

Howard Lee Mohn

September 1994

Thesis Advisor: David R. Pratt

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave Blank)

2. REPORT DATE
March 1994

3. REPORT TYPE AND DATES COVERED
Master's Thesis

4. TITLE AND SUBTITLE
Implementation of a Tactical Mission Planner for Command and Control of Computer Generated Forces in ModSAF (U)

5. FUNDING NUMBERS

6. AUTHOR(S)
Mohn, Howard L.

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/ MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING/ MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the United States Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE
A

13. ABSTRACT *(Maximum 200 words)*

The purpose of this work is to develop a three-level architecture for mission planning and task assignment to computer generated forces. This architecture is based on the Rational Behavior Model, which was constructed by Byrnes, et.al. as a means of mission planning and control for autonomous robots. Extending this concept to address the problems of mission planning for computer generated forces allows the human greater flexibility and capability in controlling large numbers of computer generated forces in a large-scale virtual environment.

The base system used in this proof-of-concept prototype is the Modular Semi-Automated Forces system (ModSAF), which was developed by Loral ADS for US Army Simulation, Training, and Instrumentation Command, and written in C, using OSF/Motif as the graphical user interface (GUI) system. A prototype mission planner was added as a library to this application, using the US Army's five paragraph operations order as the basis for a series of GUI editors. The editors provide information to the framework about which artificial intelligence modules operate on the data input from the order, generating ModSAF tasks that are subsequently executed by the company. Currently, the input is parsed directly into a series of company-level ModSAF mission tasks.

The initial results from the prototype resulted in a significant simplification of task generation for the user. One operations order phase generated on the average 2.5 ModSAF phases, with no requirements for additional parameter changes. Further research is needed, however, to fully determine the resource implications of including AI modules in an already complex system. The use of the operations order as a means to generate a company-level mission simplifies mission generation, but a robust expert system is needed to effectively convert the operations order input data to a set of ModSAF tasks.

14. SUBJECT TERMS
Computer Generated Forces, Command and Control, Mission Planning, Distributed Interactive Simulation

15. NUMBER OF PAGES

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	Unlimited

Approved for public release; distribution is unlimited

IMPLEMENTATION OF A TACTICAL MISSION PLANNER FOR COMMAND AND CONTROL OF COMPUTER GENERATED FORCES IN MODSAF

by

Howard Lee Mohn
Major, United States Army
B.S., Texas A&M University, 1980
B.S., University of Maryland, 1992

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL

September 1994

Author:

_____

Howard Lee Mohn

Approved By:

_____

David Pratt, Thesis Advisor

_____

Robert McGhee, Second Reader

_____

Ted Lewis, Chairman,
Department of Computer Science

ABSTRACT

The purpose of this work is to develop a three-level architecture for mission planning and task assignment to computer generated forces. This architecture is based on the Rational Behavior Model, which was constructed by Byrnes, et.al. as a means of mission planning and control for autonomous robots. Extending this concept to address the problems of mission planning for computer generated forces allows the human greater flexibility and capability in controlling large numbers of computer generated forces in a large-scale virtual environment.

The base system used in this proof-of-concept prototype is the Modular Semi-Automated Forces system (ModSAF), which was developed by Loral ADS for US Army Simulation, Training, and Instrumentation Command, and written in C, using OSF/Motif as the graphical user interface (GUI) system. A prototype mission planner was added as a library to this application, using the US Army's five paragraph operations order as the basis for a series of GUI editors. The editors provide information to the framework about which artificial intelligence modules operate on the data input from the order, generating ModSAF tasks that are subsequently executed by the company. Currently, the input is parsed directly into a series of company-level ModSAF mission tasks.

The initial results from the prototype resulted in a significant simplification of task generation for the user. One operations order phase generated on the average 2.5 ModSAF phases, with no requirements for additional parameter changes. Further research is needed, however, to fully determine the resource implications of including AI modules in an already complex system. The use of the operations order as a means to generate a company-level mission simplifies mission generation, but a robust expert system is needed to effectively convert the operations order input data to a set of ModSAF tasks.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENT

Numerous individuals provided me invaluable assistance to conduct this research. First, I wish to thank Professor David Pratt and Professor Robert McGhee for providing me the guidance and inspiration to study autonomous force development topics. Second, I wish to thank Joshua Smith for his assistance in helping me to understand the ModSAF source code. He also modified one of the ModSAF libraries to support my research, and patiently answered every one of my many questions. I received additional help and guidance from Dr. Andy Ceranowicz, Marnie Salisbury, Dr. Se-Hung Kwak, Anthony Courtemanche, and Thomas Stanzione. To these individuals, and the others with whom I discussed my ideas, I owe a debt of thanks.

Finally, I wish to thank my wife, Lisa, and our five children: Shibahn, Gweneth, Sarah, Karl, and Alec, for their support and patience during the last two years. Their love provided me the motivation to complete this work.

I. INTRODUCTION

A. MOTIVATION

The United States Army, as well as the other branches of the armed forces, has committed itself to the development of virtual world technologies and distributed interactive simulation (DIS) as a means to effectively and economically train their forces. The use of DIS could mean that units would no longer be required to travel to training areas as often as they do now, and would reduce the wear and tear on military equipment, not to mention the reduced effects of environmental damage. Individual soldiers, crew members, and unit leaders would all benefit from real-time, realistic environments, allowing them to practice much more frequently their essential combat skills. Today, DIS has proven itself effective in networking small numbers of simulators to train platoon-sized (four vehicles) elements and individual vehicle crew members.

The Army's vision is to have the ability to run large-scale, virtual simulations with many thousands of entities by the end of the 20th Century. To accomplish this feat, the number of available simulators must grow dramatically in the next few years. The total number of manned simulators in a virtual battlefield, however, would be far less than the number of computer-generated entities, because the cost of constructing and manning individual simulators is significantly more expensive. For training to be effective and realistic, these computer-generated forces (CGF) must be endowed with an awareness of their environment, their capabilities, and a fundamental set of behaviors that would allow them to realistically portray a vehicle or entity on the battlefield. In the best case, CGF and manned simulator entities should be indistinguishable [PETTY94].

B. PROBLEM

If one is to use CGF effectively, then an economy of human interaction is needed. The use of CGF today is somewhat limited because of the inherent complexity in modelling the behaviors of a realistic force. As a result, most CGF applications are limited to the repre-

sentation of individual vehicle/person behaviors. In this approach, the human SAF controller is responsible for the unit-level behaviors and the inter-vehicle interactions and coordinations.

Other CGF applications model low-level unit behaviors and tasks, but cannot perform mission analysis and planning. In addition, these applications require the human SAF controller to closely monitor the scenario actions, and be prepared to make timely decisions to directly intervene in the simulation process. Failure to intervene usually results in nonrealistic behaviors at the unit (aggregate) level.

One of the latest examples of this type computer generated force applications is the Modular Semi-Automated Forces system (ModSAF), developed by Loral Advanced Distributed Simulation [DMSO93]. While it represents "state of the art," it has shortcomings in the requirement for the human operator to constantly intervene to "correct" inappropriate CGF behavior. In particular, each entity up to and including platoon level must be given highly detailed sets of mission tasks for it to represent a minimally acceptable level of competence. In many cases, elements of a platoon must be given separate instructions, as is the case with a mounted section and a dismounted section of a mechanized infantry platoon.

C. SOLUTIONS

There are several improvements that can be made to existing SAF systems. However, the fact that they are not yet in existence is proof that such an undertaking is not trivial. These include:

1. Define Real-Time Behavior Mechanisms

The definition of real-time behavior mechanisms to exercise real-time coordination among related entities would require aggregation in some manner of the individual vehicle/soldier behavioral actions, and an increase in the "awareness" of each individual computer generated force element of their respective mission relative to each other. Additionally, a means of cooperative communication among different CGF entities would be required.

2. Construct a Mission Planner to Simplify Mission Tasking/Assignment

This is the subject of this work. A mission planner would assist the user by abstracting the individual vehicle/platoon level tasks, allowing the user to assign tasks at a higher level. The mission planner is a first step towards the development, testing, and implementation of an artificial intelligence system that controls lower level behaviors, while the human is freed to concentrate on the aggregate behaviors.

D. FOCUS/SCOPE OF WORK

The focus of this work is to develop a proof-of concept prototype of a company-level command and control mission planner. This initial work lays the foundation for the incorporation of artificial intelligence languages to simulate a company commander's mission analysis and course of action development.

This mission planner is written for the ModSAF semi-automated forces system. The initial design considerations attempted to make the mission planner as loosely-coupled with ModSAF as possible; this was not possible given the inherent complexity of ModSAF and the emphasis on the reuse of existing ModSAF libraries and modules in the construction of the mission planner application.

The mission planner allows the user to enter a modified battalion-level five paragraph field order. The input data would be used by the various modules within the mission planner to determine the most effective course of action by the company to accomplish the battalion's mission. Its output consists of a set of ModSAF mission task frames that executes the order.

E. ORGANIZATION

A discussion of previous work related to this problem is discussed in Chapter II. It includes a relatively detailed discussion of the Rational Behavior Model and its application in autonomous robots. Current and recent research on automated mission planning is also presented.

A summary of the ModSAF architecture and design is presented in Chapter III. The information includes the essential concepts required to understand the role of the Mission Planner within the ModSAF system, and provides the contextual background for the remaining chapters.

The architecture of the Mission Planner is presented in Chapter IV. The major components of the Mission Planner are discussed, along with their relationship to each other. This provides an overview of the information flow within the Mission Planner.

Design strategies are discussed in Chapter V. Two basic design strategies are presented, along with their advantages and disadvantages. Design considerations that are common to both strategies are identified and presented.

The development of the Mission Planner is presented in Chapter VI. Both design strategies were attempted; the problems encountered for each strategy are discussed, along with observed solutions.

The results of the Mission Planner development are discussed in Chapter VII. One of the design strategies presented in Chapter VI proved to be unfeasible due to time restrictions; reasons and recommendations are discussed. These restrictions on available time prevented full implementation of a fully operational prototype system, therefore, the final proof-of-concept prototype is discussed.

Chapter VIII includes a discussion of the merits of the strategy of the overall approach, and recommendations for future research. Suggested improvements are included. The appendices provide additional information, to include a user's guide and a programmer's manual. These are meant to supplement, not replace, the ModSAF User's Guide and programming documentation.

II. PREVIOUS WORK

The development of a meta-level mission planning application for command and control of computer generated forces is a relatively new area of study. As a result, the body of available literature that directly addresses this topic is sparse. As a result, background information in three subject areas that had an impact in the development of this CGF mission planner are presented. They are: a) The Rational Behavior Model, b) automated mission planning, and c) autonomous force collaborative reasoning and belief systems.

A. RATIONAL BEHAVIOR MODEL

The Rational Behavior Model is a three-level software architecture proposed for use in autonomous robots [BYRN93]. It proposes a separation of the software control modules into three, separate, and functionally different areas. This allows for an efficient means to link high level symbolic computations to low level control software. The three levels are called the Strategic, the Tactical, and the Execution levels of abstraction.

1. Strategic Level

The highest level of abstraction is the Strategic Level. This level allows the user access to an effective means of expressing a mission to the vehicle. As such, it contains high-level logic that is required to control a robotic platform and the mission it is to perform. This level consists of a top-down, goal driven approach to the mission, which lends itself in implementation to a symbolic, asynchronous, discrete (Boolean) domain. Byrnes, et.al. recommended the use of Prolog as a specification language, with a form of conversion to another symbolic language (such as Lisp), or to a procedural language such as C or C++ [BYRN93].

When implemented in Prolog, mission execution can be specified in the form of a depth-first search of a dynamic AND/OR graph. Thus, goals are reached when the subgoals that comprise those goals are reached. This recursive decomposition of the goal state into its constituent subgoals is consistent with high level human mission planning.

2. Tactical Level

This level performs the required interface between the symbolic, abstract, knowledge-based Strategic Level and the individual robotic control subsystems of the Tactical Level. Byrnes, et.al. implemented this level in an object-oriented language as a set of objects and the methods that operate on those objects. The tactical level performs two functions:

a. Implement Subgoals Specified by the Strategic Level

The Tactical Level responds to the Strategic Level by implementing a set of specified subgoals. By nature, these subgoals are data-driven, forward chaining search spaces. Thus, the tactical level implements a set of "drills" that are essentially iterative in nature by determining which discrete events to execute over continuous time.

b. Manage and Translate Information Between the Strategic and Execution Levels

Since the Strategic Level operates in a discrete space independently of time, the Tactical Level must have the capability to react to the commands and information from the Strategic Level at any time. On the other hand, the execution level communicates with the Tactical level on a timed interrupt basis. Thus, the Tactical Level monitors the status of the subgoals specified by the Strategic Level, and reports their status based on feedback from the Execution Level.

3. Execution Level

This level controls the actual motors, sensors, and control surfaces in a robot. It responds to particular commands from the Tactical Level and reports the status of the command through interrupts to the Tactical Level. As implemented in the NPS Autonomous Underwater Vehicle [BYRN93], this level consists of the set of functions that provides electrical control and feedback to the control surfaces.

B. MISSION PLANNERS

While much work has been done on automated mission planners, the applicability of these planners to command and control of computer generated forces are still in the concept/prototype stages. Most of the existing planners for military use are abstracted to a much higher level than required for intermediate to low level command and control.

An example of such a planner is Eagle-AP, developed by Mitre Corporation [SALI93]. This planner requires a significant amount of data preprocessing, and is best suited for development of a brigade and division level command and control system, for which it was designed. Eagle-AP is a state-based search planner, using the concept of adversarial planning to form counter-options to the anticipated moves of the antagonist.

Other proposed systems include the use of simulation to conduct mission planning, and the use of an Artificial Intelligence blackboard to serve as a central repository for Command and Control-related messages [LEE94][BRAU93].

C. OTHER AUTONOMOUS FORCE SYSTEMS

Autonomous forces and computer-generated force systems have been in existence for some time. Examples of some mature, yet active, systems include Janus-A, the Simulated Warfare Environment Generator (SWEG), and Battalion/Brigade Battle Simulation/Distributed Interactive Simulation (BBS/DIS) [DMSO93]. Some of these systems, such as Janus-A, are undergoing modification to allow them to interact in a Distributed Interactive Simulation (DIS) environment, while others are fully DIS-compliant.

1. Autonomous Forces in NPSNET

A three-level system of collaborative intelligent CGF agents was proposed by Pratt, et. al. [PRATT94]. In this system, autonomous force consists of three basic elements: an observer, a decision-maker, and an execution agent. The decision-making module is further broken-down into three distinct levels of responsibility: individual, crew, and unit. The individual level models the immediate actions required to complete an assigned task. It also is the level at which the overall decision-making module interfaces with the execu-

tion agent. The crew level models the actions required by a vehicle crew, in this case, an M1A1 tank. This level can be best described as a collaborative effort, in which vehicle tasks such as movement, turret traversal, and target acquisition are all determined by the crew commander. Finally, the unit level models the actions of a unit leader in coordinating the assets available to best accomplish the mission. There are as many unit level modules as there are units in the hierarchy. This is the level where mission planning, unit task selection, and timing of actions occur. The advantage of this concept is that meta-level reasoning can be employed at the crew and unit levels of decision making. Actions selected by the higher level unit are transmitted to the lower level unit/crew in the form of tasks/ directives that the lower level unit incorporates into its reasoning system. Thus a coherent form of hierarchical command and control can be developed.

2. Use of Fuzzy Logic for CGF Behavior Representation

Fuzzy logic has been proposed as a decision-making system for high-level command objects in the Maneuver-Warfare Analytical and Research System currently under development for the US Marine Corps [PARS94]. This decision-making model would imitate a brigade-level commander and higher, focusing on the key relationships between a real-life commander and his supporting staff. All reports (input) are "fuzzified," and the commander makes decisions based on the values in the fuzzy sets. The final decision is computed by finding the centroid value through the process of "defuzzification" [DURK94]. The advantages of the use of fuzzy logic in military decision making include expressing the situation in terms of linguistic variables, making fuzzy rules easier to understand, and the fact that most battlefield information is by nature a fuzzy value ("fog of war").

D. SUMMARY

Mission planning for computer generated forces is an evolving field of research. Prototypes have been constructed at the battalion level and higher; the recent developments in lower echelon modeling and simulation are generating a need for the capability of these

agents to plan and coordinate with each other as well as react to changes in their environment. The Rational Behavior Model is a method of high-level control of autonomous robots; its principles could be applied to command and control of computer generated forces as well. In the next chapter, a fairly detailed overview of the Modular Semi-Automated Forces system is discussed. This will provide the contextual framework around which the Mission Planner is integrated.

III. MODSAF DESCRIPTION

The Modular Semi-Automated Forces (ModSAF) system is a set of software modules that replicate the presence of simulated vehicles and their associated equipment in a Distributed Interactive Simulation (DIS) environment [LORAL94]. The following is a brief overview of the system; the ModSAF architecture is described in detail in [LORAa93].

A. SYSTEM OVERVIEW

ModSAF consists of three basic application programs: the *SAFstation*, the *SAFsim*, and the *SAF Logger*. These programs interact with each other on a network by sharing a common database -- the *Persistent Object Database* [LORAa93]. These programs communicate with each other through two different network protocols: the DIS (or SIMNET) protocols, and the internal Persistent Object (PO) Protocols. For small simulations and testing purposes, the SAFsim and SAFstation can be run on the same workstation, without the requirements of the network protocols.

A ModSAF Suite can be run in any combination of SAFstations, SAFsims and Loggers. As a general rule, fairly large simulations will require the use of one SAFstation, and several SAFsims, with an optional Logger, as shown in Figure 1. The ModSAF Suite at the Naval Postgraduate School consists of five Silicon Graphics Iris Indigo XS 4000 workstations, with a MIPS R4000 processor, 80 Megabytes of Random Access Memory, and a 1.2 Gigabyte hard drive [SILIC92].

1. SAFstation Description

The SAFstation consists of the graphical user interface that allows a human operator to generate and interact with the computer generated forces. It consists of a map display and a series of editors and tools to facilitate construction of mission scenarios and also serves as a means for entering commands to the system. Thus, the SAFstation is the means

to set up and control simulation exercises, in addition to modifying the missions and behavior of existing computer-generated forces.

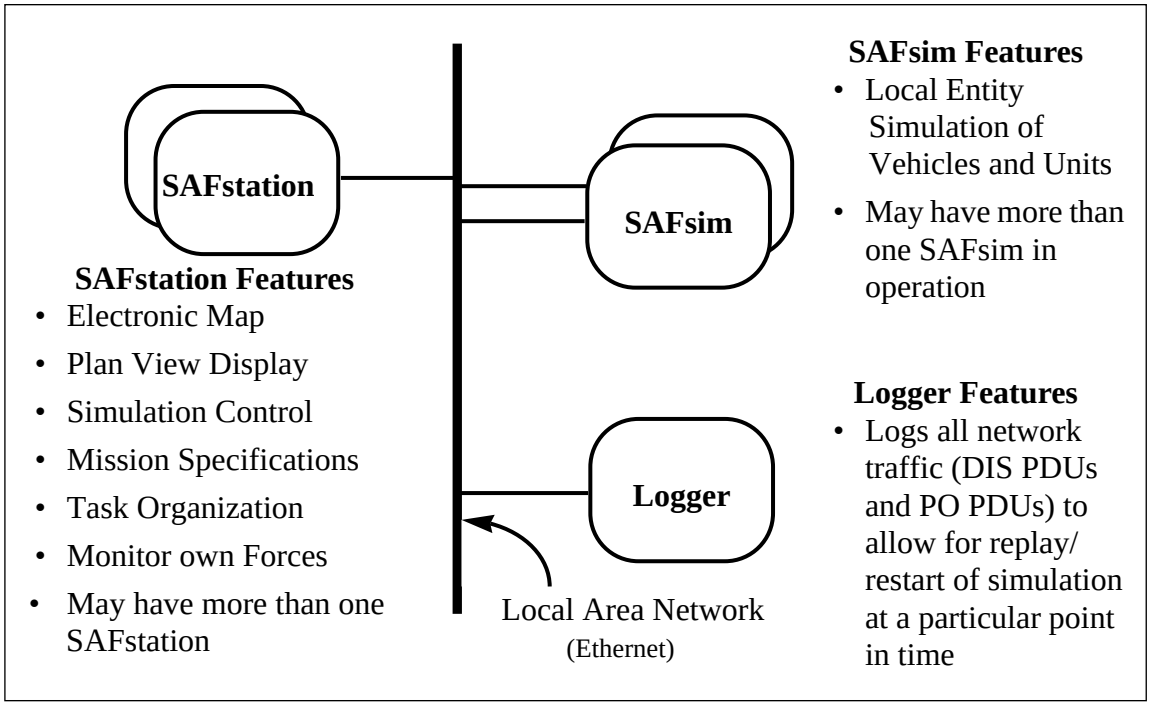


Figure 1: ModSAF Architecture, after Ref. [LORAb93]

The SAFstation features a graphical user interface consisting of a 2-dimensional map display[1] and a series of tool buttons to allow the user to examine the terrain, monitor the tactical situation, and prepare orders. A message log records commands issued and the reports flowing back. The SAFstation performs no simulation; it issues requests for action to the SAFsims through messages and orders, making heavy use of the PO database. The purpose of separating the orders and Command and Control (C2) functions from the simulation functions is to facilitate the substitution of other types of SAFstations, such as an artificial intelligence overlay, and to allow the SAFstation to run on different platforms. [CERb94][LORAL94]

1. Also called the Plan View Display (PVD)

2. SAFsim Description

The SAFsim is the simulation engine of the ModSAF suite. As such, it simulates both collective and individual behaviors, in addition to the interactions of the CGF to each other.

3. SAF Logger Description

The SAF Logger provides a means to record and play back simulation exercises conducted on the virtual battlefield. The logger records both computer generated and manned simulator entities. It also can record the command and control information that flows between the other SAF subsystems. As such it could provide a more detailed analysis of SAF behavior during construction of new tasks and missions. [LORAa93]

B. PERSISTENT OBJECT DATABASE DESCRIPTION AND FUNCTIONS

A critical element of the ModSAF system is the Persistent Object (PO) Database. This unique approach to the critical task of state information management for multiple entities was dictated by the distributed nature of the system, and the need for all elements of the system to have universal access to that information. [CALD93][CERb94][CER92]

The sharing of vehicle appearance data among subsystems is accomplished through the DIS protocol. However, there is a great deal of command and control data which cannot be shared using the DIS protocol. The sharing of C2 data in ModSAF is accomplished through the Persistent Object protocol. This protocol defines the classes of objects which can be shared between simulations running on separate hardware platforms. Software functions are included to allow simultaneous editing of objects, and to ensure persistence of objects despite hardware failures. The C2 data is used to update the PO database contents to reflect the current state of the simulation. [CALD93]

Persistent Object Protocol is similar to the DIS protocol in that it is broadcast through the network for all ModSAF stations. It has seven basic protocol data units to handle the C2 data transfer: Simulator Present, Object Present, Delete Objects, Nomination, Describe Object, Object Request, and Set World State. The PDUs can perform load balancing among

more than one SAFsim station, and handle the migration of vehicle task controls between hosts. [CALD93]

As its name implies, the Persistent Object database is object-oriented. Table 1 provides a list of persistent objects and their definitions:

Name	Definition
Control Measure	Point, line, area or route
Unit	Entity or unit
Task	Individual behavior. Examples include moving object collision avoidance, platoon bounding overwatch
Task Frames	Collections of tasks which execute in parallel and form a component of a mission
Task Frame Stack	Collection of task frames which a unit is currently executing. Only one task frame is active at a time
Missions	Represented by a network of task frames
Overlays	Organize persistent objects representing orders of battle, intelligence information, missions
H-Hour	Used to synchronize mission execution by different units

Table 1: Persistent Objects, from Ref. [SMITHa93]

The PO database is replicated on all SAFstations and SAFsims running the same simulation. Modifications to the PO database require a near-simultaneous update by all workstations. This is accomplished through a two-step update process. Both SAFstations and SAFsims modify the database contents as the situation warrants. However, they do not react to the modification until the change is returned via an event. Thus, the event is broadcast to all ModSAF elements in a manner similar to DIS broadcasting. The event signals all SAFstations and SAFsims that a change has occurred to the PO database, thus ensuring that all stations receive the change at the same time. [CER92][SMITHa93]

C. ASYNCHRONOUS AUGMENTED FINITE STATE MACHINES

The software implementation of a task is through a finite state machine. The task is thus represented as a series of actions to take given a set of inputs. In this case, ModSAF uses an Asynchronous Augmented Finite State Machine (AAFSM) to represent a particular task. The state machine is asynchronous because it may generate outputs in response to events in the simulated environment, not based on a time function. It is augmented because it can influence and use many variables other than its own state variable [CER92]. The AAFSM and the data structures it operates on are called a task model.

A task model interacts with four separate data structures. Some of these are shared and exist in the PO database where any ModSAF subsystem can examine and manipulate them. Others are local, meaning they are only available to the simulation process that is modelling the vehicle performing the task. The data structures can be either public or private. The four data structures are:

1. System Parameters

These are shared, public parameters which control how a task performs its job. These parameters are set for each vehicle platform type to specifically guide the execution of the general task in a manner appropriate to that platform. (i.e. tanks do not fly, etc.) These parameters are also used to tune execution, and to allow modification through the PO database. [CALD93]

2. Task Parameters

These are shared, public parameters which control the execution of a task in the context of a single mission. These include mission parameters such as which route to follow, and rules of engagement. [CALD93]

3. Shared Task State

This is a shared, private state of an executing task. It includes the AAFSM state machine variable and any other specific state information which needs to be shared for use

by other programs executing in the ModSAF system. This state should not change very frequently, as each state change causes additional network traffic. [CALD93]

4. Local Task State

This is a local, private state of an executing task. It usually contains more detailed information than the shared task state, and will represent the same information in the shared task state in a more efficient manner. This state may change frequently without any network overhead, but this updated information is not available to any other programs in the ModSAF system. [CALD93]

D. COMMAND AND CONTROL ARCHITECTURE

ModSAF uses a hierarchy of tasks, task frames, and a frame stack for each unit/entity to simulate that unit's behavior in the virtual world. The developer's stated objectives in this area are:

- to support complex missions, including preplanned contingencies and task reorganization,
- to issue a fragmentary order (FRAGO), which interrupts and immediately changes the current mission,
- to override choices made by the simulated units at run time,
- to provide a succinct method of representing missions,
- to provide a general representation for unit and individual behavior within the architecture without mandating a specific approach to behavioral representation,
- to provide a method for representing battlefield uncertainty,
- to provide a user interface which is constructed automatically from the software definitions of available missions,
- to provide the ability to record command and control information and then subsequently restart from any point in the recorded exercise,
- to provide a structure which allows the user interface software to explain unit and individual behavior to the user, and
- to provide a software architecture which allows new behaviors to be encoded in any language -- language independence. [CALD93]

1. Tasks

The basic building block of the ModSAF command and control architecture is the task. There are five types of tasks in the ModSAF system: unit tasks, individual vehicle tasks, reactive tasks, enabling tasks, and arbitration tasks. A task is represented in the PO database by a unique task model number, the task parameters, and the task state. All tasks except the arbitration task are represented as an AAFSM. [CALD93][CER92]

Unit tasks model behaviors which are performed at a unit level. In this case, a "unit" is any military organization larger than individual vehicle, and smaller than a battalion (i.e. platoon and company). These tasks will create, delete, and monitor the progress of lower-level tasks for subordinate units and vehicles. The hierarchical manner of military organizations and decision making is thus preserved. Examples of unit tasks are Company Road March, Platoon Bounding Overwatch, and Company Attack. [CALD93]

Individual vehicle tasks model behaviors that an individual vehicle would typically execute. These tasks will control and process information from the physical subsystems of the vehicle (e.g. weapons status, turret direction, etc.). Examples of individual vehicle tasks are Follow Route, Keep Formation, Avoid Collisions, and Spot Enemy Vehicles. [CALD93]

Reactive tasks are used to trigger reactions to events in the virtual battlefield which may be encountered by a unit or a vehicle. These tasks are pre-defined by the ModSAF system and stored as data files. At run-time, the user can enable and disable them, or modify the parameters to meet the requirements of a particular mission. Examples of reactive tasks are Air Raid Happening, Target Meets Commit Criteria, and Hasty Attack Needed. [CALD93]

Enabling tasks model behaviors which will trigger mission contingencies. They are defined by the user during the construction of a mission, and allow the user to specify alternative actions for a unit to take in response to conditions or events which the user predicts may occur during the mission. Examples of enabling tasks are Crossed Phase Line, Detected Enemy Unit, and Reached H-Hour Time. [CALD93]

Arbitration tasks are a special class of tasks which take a set of multiple, competing recommendations from tasks and form a single recommendation. These recommendations may be at any level of the hierarchy from individual vehicle through unit (company and higher). Examples of arbitration tasks are Vehicle Movement Arbitration, Vehicle Sensor Arbitration, and Vehicle Targeting Arbitration. [CALD93]

2. Task Frames

Task frames are used to group a collection of related tasks which run in parallel. Each task frame represents a mission phase, such as Occupy Assembly Area, Road March, or Attack. Task frames are defined in a data file by a set of task names and the task parameters associated with each of those tasks. Some of the parameters can be modified by the user, allowing the task frame to be customized for a particular mission. [CALD93]

Task frames can either be constructed by the user or generated internally by the simulation software as it executes. At the beginning of the simulation, the user will generally specify one or more task frames to construct a mission for a particular vehicle or unit. The simulation software will construct its own task frames in order to simulate the downward flow and refinement of commands, or to respond to certain events on the virtual battlefield. [CALD93][CER92]

Task frames also support the hierarchical structure of military organizations in that they can be defined for both individual vehicles and units. To do this, a ModSAF vehicle must be able to perform multiple “roles” in the unit hierarchy (This is congruent to actual military practice, where a tank platoon leader is also the commander of his individual tank). Therefore, a single vehicle can have multiple task frames -- some to perform unit level missions, and others to perform the individual vehicle behaviors.

Unit task frames are congruent to unit tasks -- they model the activities which a unit leader performs on the battlefield. They are composed of unit tasks, and may add, modify, and delete tasks and task frames in subordinate units and vehicles to control their behavior.

Individual task frames are defined to model the activities which a single vehicle performs on the battlefield. They are composed of individual vehicle tasks which perform such functions as moving, shooting, target acquisition, etc. Each vehicle has at least one task frame where its tasks reside. [CALD93][CER92]

3. Task Frame Stacks

There can occur situations in any battle, simulated or otherwise, when a mission is temporarily suspended to react to a more critical event -- one which would prevent mission success, or one which exploits a particular enemy vulnerability. This is modelled in ModSAF by the implementation of task frame stacks.

Rather than execute a single task frame, ModSAF units execute a stack of task frames. The topmost frames are currently active; the rest are suspended. New task frames may be pushed on the stack as the result of a reactive task, or at the direction of the user to perform some immediate action. When a task frame has been completed, it may be popped off the stack to resume a previous activity, or a new task frame may be pushed on the stack to initiate a new activity. [CALD93][CER92]

Each task frame in the stack is marked as transparent or opaque. If the topmost task frame is transparent, then its tasks are merged with the tasks of the topmost opaque frame. Multiple transparent frames may be layered in this manner to avoid unnecessary duplication of unaffected tasks in the top frames. Thus, minor modifications to the mission and mission overrides can be efficiently handled through the placement of transparent task frames on the task frame stack. These transparent frames will contain only the tasks that are affected by the modifications. When the mission is to be resumed, these transparent task frames are simply removed from the task frame stack. [CALD93][CER92]

4. Missions and Enabling Tasks

A mission is a collection of task frames which are linked together to form a sequence of operations. In a simple mission, each task frame specifies one task in the previous task frame which must end before the subsequent task frame starts. For example, if a unit

should perform a road march along Route A to Release Point P, and then attack along Axis B, then the attack task frame will not start until the road march task frame ends. The road march task frame signals its completion when the unit arrives at Release Point P. [CALD93]

In more complex missions, the task frames are linked together by enabling tasks in addition to the completion of executing tasks. Enabling tasks are constructed to execute based on certain conditions occurring, such as the spotting of a particular type of enemy unit in a particular location. The enabling tasks link together the task frames which make up the various contingencies of a mission, as defined by the user. [CALD93]

5. Task Manager

The task manager is a portion of the command and control database software that determines the outcome of the recommendations from the task AAFSM operations. As such, it is responsible for the overall implementation of the C2 architecture. When Mod-SAF is started, each task model registers itself with the task manager. Each model specifies the task model number, the entry points for the task functions which the task manager may need to invoke, the task shared-data size, and a list of other task models which must run before and after that particular task model. The "before" and "after" lists are essential to determine the task dependencies to the task manager. [CALD93][CERb94]

Once this information has been provided by each task model, the task manager uses the following algorithm to determine which tasks run and when:

1) Get a list of roles which this vehicle is performing (such as company commander, platoon leader, etc.). One of these roles is always that of the vehicle itself.

2) For each role, determine if any of the current task frames have been unassigned. If so, remove those task frames from the stack.

3) For each role, determine if the enabling tasks of any subsequent task framed indicate that those frames should be executed. If so, start the new frame.

4) For each role, traverse the current frame and create a list of tasks to execute.

5) Sort the task execution list such that the “before” and “after” constraints of each task model are satisfied.

6) Examine the resulting task execution list to determine if any tasks which were executed last simulation clock “tick” are no longer in the list. If any tasks meet this criteria, then suspend them.

7) Execute the tasks in the task execution list in order. [CALD93]

The task manager also handles the roles of unit leader and individual vehicle task frame stacks. It handles the pushing and popping of task frames on the task frame stack. This includes the starting, stopping, and suspension of tasks within that task frame.

6. Task Arbitration

The C2 architecture is not a simple one. Since each entity, vehicle, and unit has a task frame, each with a set of tasks operating in “parallel,” there must exist a means for conflict resolution among competing recommendations for action that are formed as a task executes. Eventually, one single decision for action must result based on the task recommendations. The body of software that is responsible for deconfliction of competing recommendations to form one decision is the task arbitrator. [CALD93]

The designers of ModSAF chose a “context-rich” form of arbitration. This means that each task determines its preference, and expresses it in a way which provides the context to make that decision. An arbitrator then selects a single preference, or merges the multiple preferences using whatever context is necessary, and makes a more informed decision regarding its subsequent actions. [CALD93][CER92]

In the ModSAF implementation, each task in the current frame executes and may generate a recommendation for one or more *actuators*.[2] This recommendation is set locally, within the context of each task frame. The arbitrators, which are tasks, are executed last by the task manager. Each arbitrator will read the recommendations for each of the actuators it is responsible for controlling from each task. The arbitrator uses a selection algorithm to

2. An actuator is a command to control a physical vehicle subsystem, such as traverse a turret to a target, or move at a certain speed at a particular heading.

make a decision among the various tasks' recommendations. Currently, it selects the recommendations of the task with the greatest priority. The arbitrator converts these recommendations into commands which the actuator understands, and then sends those commands to the actuator. This is illustrated in Figure 2. [CALD93][CER92]

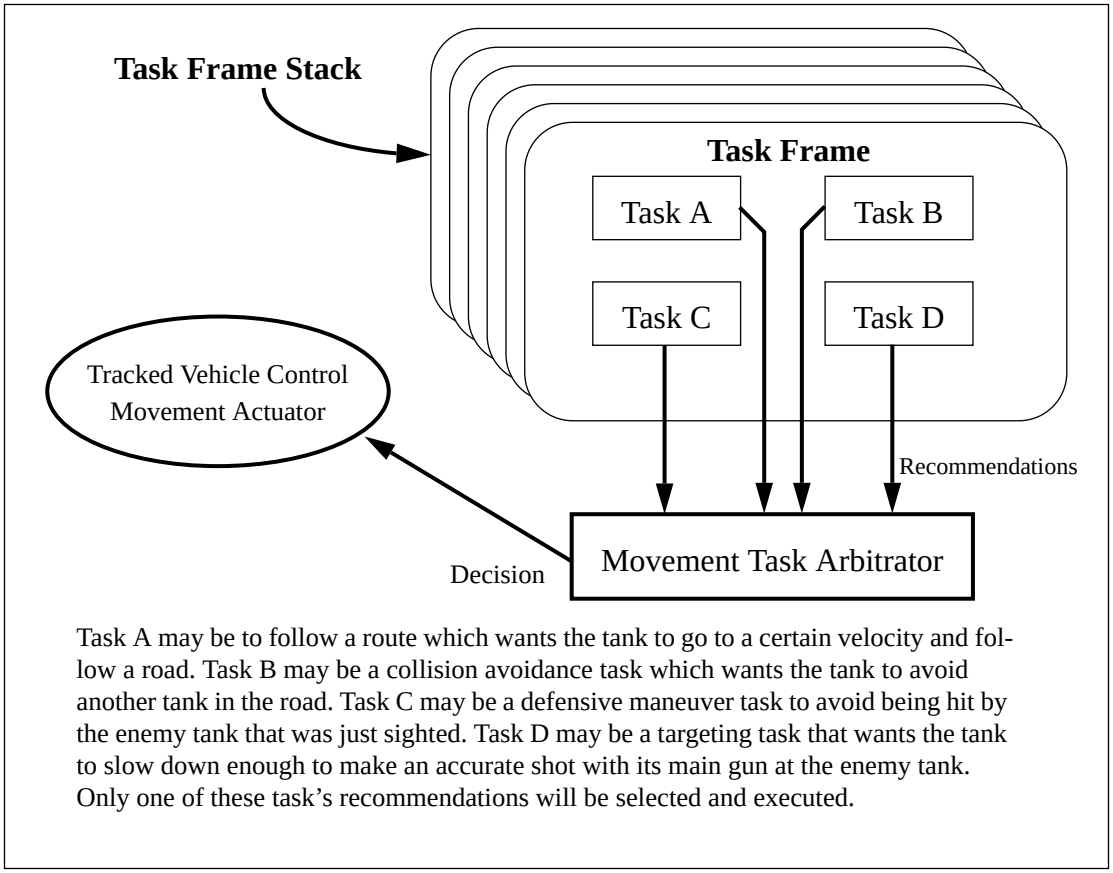


Figure 2: Tasks and Arbitration, after Ref. [CER92]

E. TERRAIN DATABASE

The terrain database is a direct derivation from an earlier computer generated forces program developed by Bolt, Beranek, and Newman (BBN) -- the ODIN Semi-Automated Forces system. It comprises two major databases: the compact terrain database, and the quadtree database. These databases are generated off-line using BBN's S1000 terrain database generation program, which takes raw data from various sources, such as the Defense

Mapping Agency's (DMA) Digital Terrain Elevation Data (DTED) and other DMA products, geological maps, photographic data, and Air Force terrain photography. [STAN93]

1. Compact Terrain Database

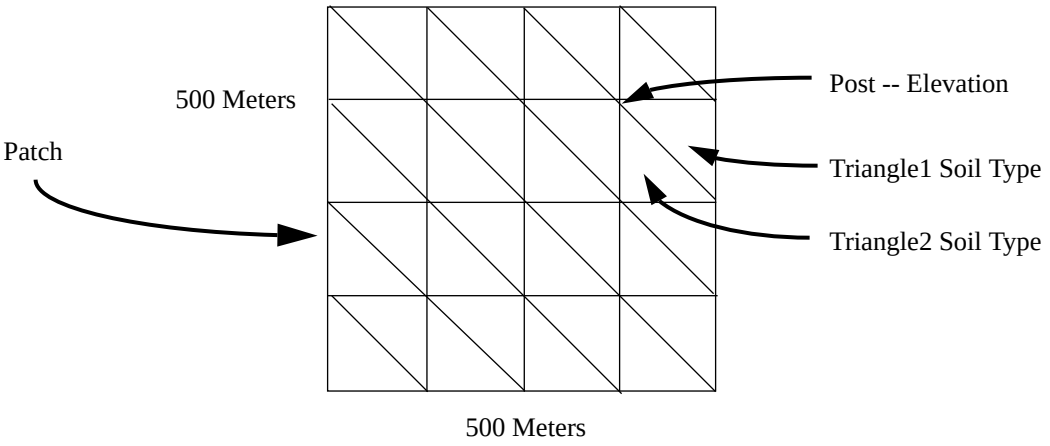
The Compact Terrain Database is derived from S1000 data and other information to generate an efficient representation of the terrain with regard to elevation, soil type, and feature data. The database is stored in vector format, which lends itself to the use of high performance algorithms for computing point-to-point visibility, radar masking, elevation lookup, vehicle placement on the terrain, and graphical display of the terrain database. The elevation and soil type data are stored in a series of posts, where each post stores an elevation and two soil types (Figure 3). Using this method, a grid the size of the Fort Knox database (50 km X 75 km) requires just under one megabyte of storage. [SMITHb93] [STAN93]

In addition to the regular grid of elevations and soil types, some areas of the terrain are modelled more precisely using microterrain. This is a collection of squares and triangles which cover a portion of a patch. This allows for the inclusion of buildings, trees, and linear features such as roads and rivers. For each patch, the microterrain and the surface features are encoded together. Table 2 shows the terrain features that are represented in the compact terrain database. The total combined elevation grid, microterrain and surface features increases to about 4.5 MB of storage requirements for the Fort Knox database.

2. Quadtree Database

The quadtree terrain database is used to represent certain terrain features as objects. This makes for a more effective structure for intelligent terrain reasoning. Features (Table 3) are represented as objects with appropriate attributes for semi-automated force reasoning. The terrain database is divided into square quad nodes, which are further subdivided down to a fixed size leaf node. [STAN93]

In Figure 4, a simple quadtree terrain representation is presented. In this example, the major quadrants are numbered as depicted. These quadrants are further subdivided into



A patch is a 4 X 4 collection of squares for a total of 500 X 500 meters.
A square is 125 meters X 125 meters, bisected by a diagonal from southeast to northwest. A post contains elevation data and soil data for the two triangles to its southeast.

Figure 3: Compact Terrain Database Format, from Ref. [STAN93]

Terrain Type	Terrain Features
Terrain Surface	Ground Polygons Water Polygons
Structures	Buildings Pipelines Power Pylons Other opaque, non-penetrable structures
Trees	Individual trees Tree lines Tree canopies
Linear Features	Roads Rivers

TABLE 2: Compact Terrain Database Features, from Ref. [STAN93]

Feature	Type	Attributes
Road Segments	Linear	Point List Width Distance Intersections Bridge X Y Extents
Road Intersection	Point	Location Segment IDs Intersections Bridge
River Segments	Linear	Point List Width Distance Intersections Fordable Bridge X Y Extents
River Intersections	Point	Location Segment IDs Intersections Bridge
Railroads	Linear	Point List Width X Y Extents
Bridges	Point	Location Point List Width
Trees (individual)	Point	Location Height X Y Extents
Treelines	Linear	Point List Height X Y Extents Impenetrable

Table 3: Quadtree Feature Attributes, from Ref. [STAN93]

Feature	Type	Attributes
Tree Canopies	Area	Point List Height X Y Extents Impenetrable Level
Buildings	Point	Location Footprint Height Type
Mobility Areas (Lakes, Oceans, Marshes, Boulder Fields)	Area	Point List Soil Type Level X Y Extents
Political Boundaries	Linear	Point List Width X Y Extents
Pipelines	Linear	Point List Width X Y Extents
Power Lines	Linear	Point List Width X Y Extents
Town Names	Point	Location Label Offset

Table 3: Quadtree Feature Attributes, from Ref. [STAN93]

.

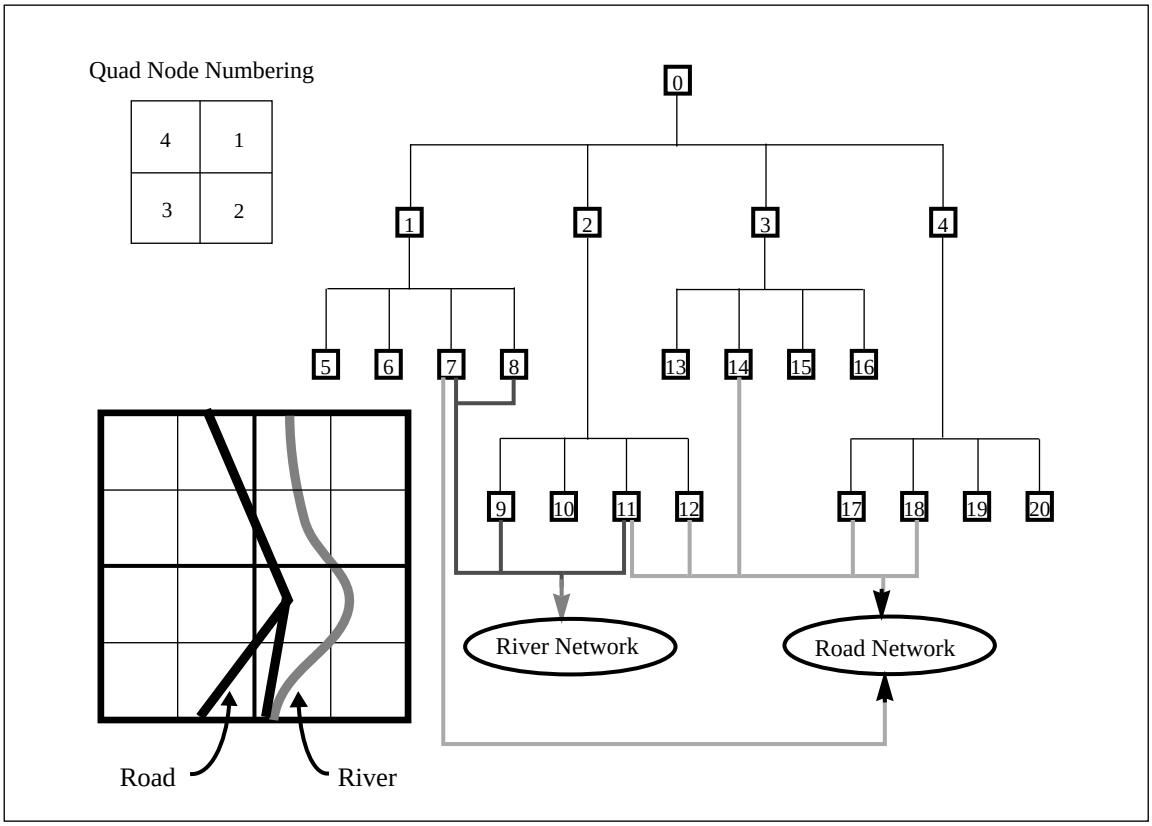


Figure 4: Quadtree Data Structure Representation from Ref. [LORAb93]

four leaf nodes each, which are also numbered in the same relative order as the parent. The sample terrain contains a river and a road, both of which are represented as linear array structures which contain a pointer to a data structure that represents the road and river network.

Each segment in the network contains such a data structure, which is essentially a record containing the midline points, width for that segment, distance, and other attributes such as a bridge (road network), or whether or not that particular segment is fordable (water network). Additionally, there is an array of pointers to a record containing a list of intersections with other segments. This record simply lists the segment ID and intersection ID of all adjacent segments and intersections. The combination of these two data structures simplifies terrain reasoning, such as moving from one point to another using a road, or in determining which route to take to successfully cross a river.

3. Terrain Database Usage

The following are the ways in which the ModSAF uses the databases.

a. SAFstation:

Uses the compact terrain database for intervisibility display tools, contour lines, and hypsometric tinting to show elevations, and a terrain cross-section display tool. It uses the quadtree database to draw the two-dimensional map display. The quadtree database is also used for route generation to create road routes (using the road network), and to check for water crossings on all ground routes (using both road and river networks). It can also generate routes around area features such as tree canopies and lakes. [CER92] [STAN93]

b. SAFsim:

Uses the compact terrain database to place vehicles on the terrain with respect to elevation and orientation. The compact terrain database is also used for intervisibility calculations between vehicles, which is required for detection and targeting. Addi-

tional uses are to calculate slope and to obtain the soil type. When flying missiles and aircraft are modeled, it is used to detect ground collisions. The quadtree database is used by the SAFsim to determine vehicle movement, especially along road routes, finding crossing points across bodies of water, and locating paths through obstacles such as tree lines, etc. Additionally the quadtree database is used to determine the relative mobility of a particular route, the effects of terrain on cover and concealment, and METT/T[3] considerations. Table 4 summarizes the way terrain is represented in ModSAF. [CER92][STAN93]

F. ModSAF SOFTWARE DESIGN

ModSAF is written in Kernighan and Richie (K&R) C with a few extensions, such as the use of longer than eight character variable names, and the definition of function prototypes. The developers used an "Object Based" design approach, meaning that subsystems are separated by the use of libraries. These libraries have a clearly defined public interface, and a private data structure that contains the information essential to the internal operation of the library. These conventions are merely an observation; the programmer could include internal variables and functions simply by including the local header file; but this is considered to be a very poor programming practice [LORAa93].

The intent behind the Object Based approach in library construction is that the public functions become the "methods" of an "object." There is a form of inheritance, as certain libraries are shared among others. For example, a turret "object" can be used by all tanks and turreted infantry fighting vehicles (IFV's); the particular characteristics of the turret are modified by the use of parameter variables. These libraries, however, do not exhibit the other traits of object-oriented languages, namely, polymorphism. Additionally,

3. METT/T: Mission, Enemy Situation, Troops Available, Terrain Considerations, Time Available -- all these are critical factors in mission planning and execution.

the incorporation of an object-oriented language, such as C++, is nearly impossible, as the main function and all function prototypes have not been defined using C++ conventions.

Feature Type	CTDB	Quadtree
Ground	3D Vectors (polygons)	not represented[a]
Trees	X, Y, Height, Width	Models[b]
Tree Lines	3D Vectors, Height	Models
Tree Canopies	3D Vectors (polygons)	Models
Structures	3D Vectors (roofline)	Models
Roads, Rivers, Rails	2D Vectors	Networks
Lakes	Part of Ground (differentiated by soil type)	Models
Bridges	Not Represented[c]	Models
Towns	Not Represented[d]	Names
Pipelines	Treated as structures	Networks
Power Lines	Not Represented	Networks
Political Boundaries	Not Represented	Networks

Table 4: Terrain Representation in the ModSAF Databases from Ref. [CER92]

a. The quadtree database can optionally load a list of contour lines, if the CTDB is not being used.
b. A model consists of a 2D outline, a height, a bounding box, a type specifier, and flag indicating whether the feature is penetrable.
c. In the CTDB, bridges occur where roads happen to cross over water.
d. The structures within a town are represented, but no grouping of these structures is done. In the sample terrain databases provided with ModSAF version C, buildings are not represented.

The current version (version 1.2) consists of a total of 596,885 lines of code and comments, divided into 255 libraries. Of this total, 396,965 lines are source code (C and in-line code); the remaining lines are comments and white space [SMITHa94]. All libraries include documentation with declaration of public functions, library requirements, and utili-

zation. Despite this, the sheer size of the code makes ModSAF extensions and enhancements a non-trivial task.

G. SUMMARY

ModSAF is a complex, distributed application that models computer generated forces at the single entity level in support of distributed simulation in virtual environments. The various programs that comprise the ModSAF system communicate with each other and with manned simulators using an agreed-upon set of protocols over a local area network. The ModSAF programs share a common Persistent Object database to distribute the work load over several systems. ModSAF is heavily data-driven; the parameters required to model vehicles and their behaviors are determined by text files that can be modified without requiring a recompilation of the program.

All entities are represented as a set of Persistent Objects. Each of these entities is aware of its environment through the execution of one or more Augmented Asynchronous Finite State Machines, which define a particular entity's behavior at a given time.

ModSAF requires frequent and detailed operator intervention for command and control of the forces. The vehicles and units will effectively react to changes in their environment, but will not transmit intentions or coordinate actions outside the unit aggregation. The next chapter presents an architecture to allow a company-sized unit to perform detailed mission planning from a battalion-level operations order generated by a human user. This mission planner is intended for pre-execution processing; this is analogous to the planning cycle used by the US Army to command and control forces on the actual battlefield.

IV. MISSION PLANNER ARCHITECTURE

The architecture of the mission planner was intended to be as modular as possible. This would closely follow the programming paradigm initiated by the ModSAF developers, and would allow for insertion of test modules for effectiveness testing. The architecture was first proposed in [MOHN94], and has evolved into its current state through implementation in ModSAF.

The architecture is as depicted in Figure 5. It consists of four distinct components: the Operations Order (derived from the Army's five paragraph operations order), the Data Formatter, the Mission Selector/Evaluator, and the ModSAF Orders Generator.

A. OVERVIEW

The mission planner can be described as a set of user-defined inputs, conversion of the input data, a reasoning process, and the conversion of the results of this reasoning to a set of ModSAF orders and phases that are assigned to the selected maneuver company. The user-defined inputs are entered via a graphical user interface, using the US Army's five paragraph operations order (OPORD)[1] format, and an associated set of overlays on a Plan View Display (PVD) (Figure 6).

The data formatter is transparent to the user, but is required to convert the input data to a form usable by the reasoning process. The reason the data formatter is separate from the data input process is to allow the employment of different reasoning processes. If the input data is relatively unchanged in scope and amount, then the only change required to incorporate a different reasoning process is the conversion of the input data.

The reasoning process (labeled "Mission Selector/Evaluator) can be any methodology that is capable of employing heuristics in determining a "workable" solution. The focus here is to optimize the planning process by constraining as many input variables as possible, yet retaining enough to derive an acceptable result. Searching for the optimal solution

1. Also known as a five paragraph field order.

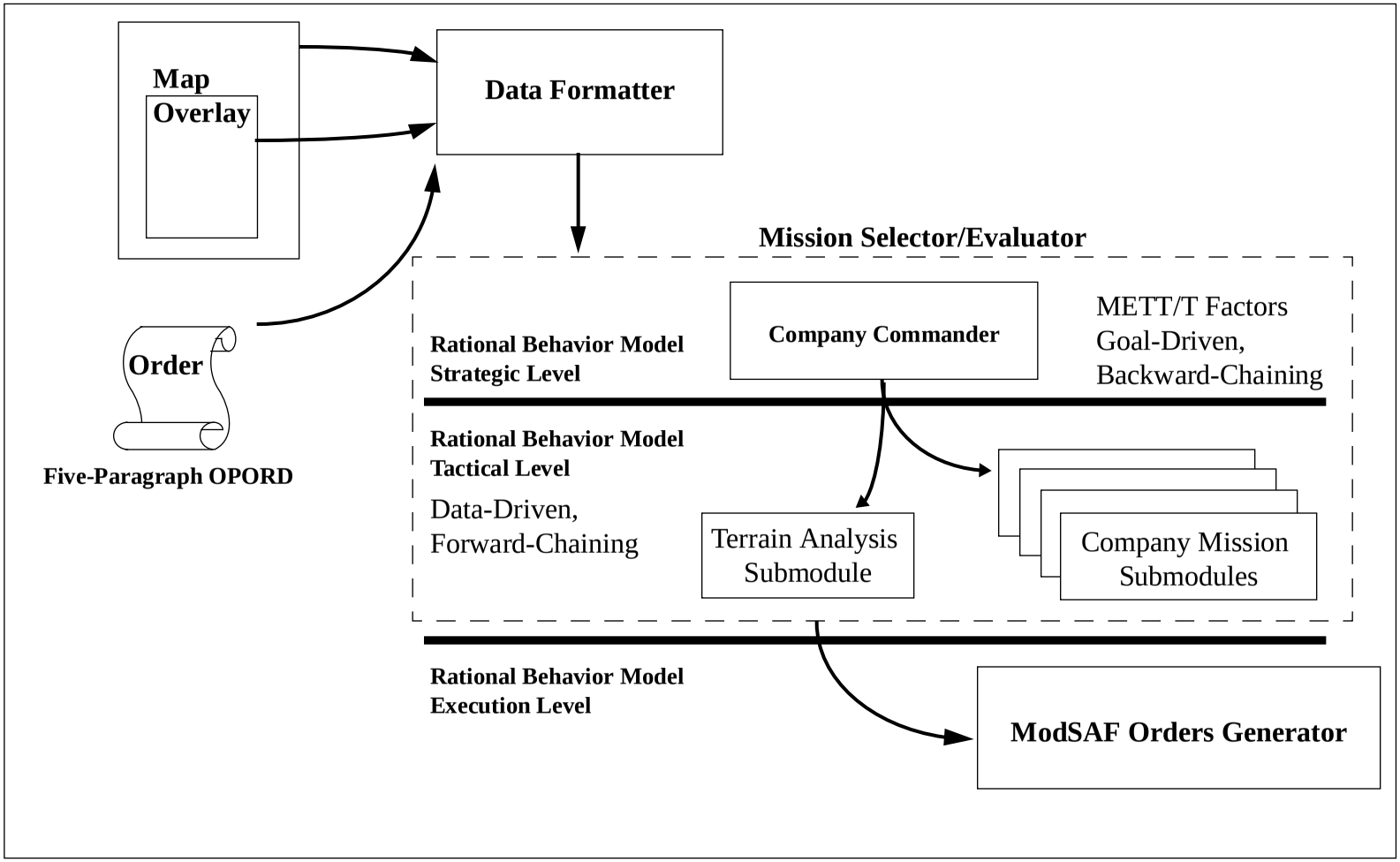


Figure 5: Mission Planner Architecture

in a given situation has been determined to be NP-complete for one prototype Multiagent Adversarial Planner [ELSA91]. It is proposed that applying the Rational Behavior Model to computer generated forces command and control will help to constrain the reasoning space, in addition to the use of heuristics. In this case, the first two levels of the Rational Behavior Model are contained within this reasoning process; the bottom level is contained within the ModSAF low-level form of control using AAFSMs and task frame stacks.

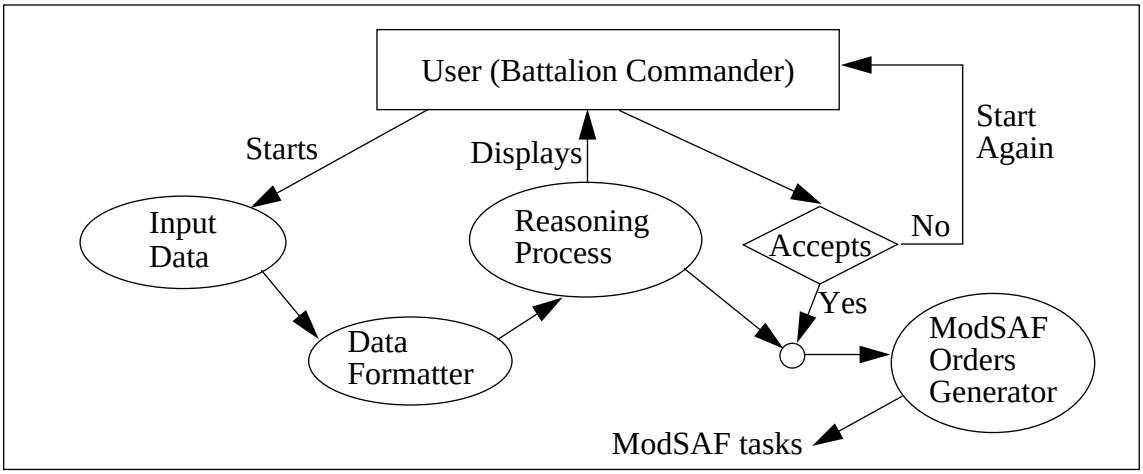


Figure 6: Data Flow Overview

The results of this reasoning process are presented to the user for approval. If the user accepts the results, then the reasoning process ends. If the user rejects the results, then the process begins anew by allowing the user to modify the input data set.

Finally, the user-approved results of this reasoning process are converted to a set of ModSAF tasks, task frames, and enabling tasks that connect them. As in the data formatter, this is a separate module that is intended to be easy to modify to accommodate different reasoning processes.

B. THE FIVE PARAGRAPH OPERATIONS ORDER

The US Army's five-paragraph operations order (OPORD) is a standardized document that enables a trained reader to rapidly develop an understanding of the overall situation, mission, commander's intent for the operation, and tasks of subordinate units. This

format is universally understood throughout the US Army, and thus is an intuitive way for a military user to assign orders to subordinate units. The five paragraphs are Situation, Mission, Execution, Service Support, and Command and Signal. See Appendix A for detailed description and sample five-paragraph OPORD.

Map overlays containing maneuver graphics are essential accessories to the basic text order. Overlays serve to graphically illustrate the contents of the order. In most cases, the OPORD text will refer the reader to these overlays, especially within the Situation and Execution paragraphs.

C. DATA FORMATTER

The data formatter takes the information that the user entered through the OPORD and the overlay and converts it to a form that is usable by the Mission Selector/Evaluator. While some forms of data require little conversion, data derived from the overlay will require processing by queries to the Persistent Object Database to derive the required information. Separation of this module from the others allows for rapid changes to be made to the data structures that result from the Operations Order module. Future versions will include a "fuzzification" submodule for conversion of input data into fuzzy sets for processing by a fuzzy logic expert system.

D. MISSION SELECTOR/EVALUATOR

This module contains the artificial intelligence submodules required to develop courses of action (CA) that meet the requirements of the operations order. These submodules comprise the first two levels of the Rational Behavior Model.

1. Strategic Level

The strategic level of the Mission Planner controls the actions of the tactical level. It will seek to fulfill the goal conditions of the order by the generation of one or more courses of action. These courses of action would be evaluated based on success expectations and ordered as such ("best" expectation first). They are then presented to the human

user for approval. If the human user approves a course of action, it is converted to a ModSAF order for subordinate units of the company.

The strategic level would take the information in the Operations Order that details the desired end state for each major phase in the operation. The user input would provide constraints to the methods available to accomplish this end state. This level is a goal-driven, backward-chaining submodule that identifies the intermediate steps required to attain the goal. This submodule would execute exactly once for each course of action to be generated.

If written in Prolog, which is a natural choice for this level, the code would look something like the following:

```
goal :- attack.
goal :- defend.
goal :- move.
goal :- assembly_area.

/* ...  */
/* The following is the source code for the “attack” subgoal tree. */
attack :- consolidate_on_objective.
consolidate_on_objective :- assault_objective, enemy_can_be_destroyed.
enemy_can_be_destroyed :- good_odds.  /* (external computation) */
assault_objective :- attack_position, company_intact.
attack_position :- axis_of_advance, company_intact.
attack_position :- last_phase_line, company_intact.
axis_of_advance :- move, company_intact.
last_phase_line :- move, company_intact.
company_intact :- company_effective. /*  (external computation) */
move :- company_deployed.
company_deployed :- wedge_formation.  /* (external computation) */
company_deployed:- vee_formation. /* (external computation) */
company_deployed:- line_formation /* (external computation) */
/* -- end -- */
/* Initial conditions:  wedge_formation, company_effective, good_odds.
 */
```

2. Tactical Level

The tactical level of the Mission Planner executes the company drills such as “move,” “attack,” and “defend.” For each of the subgoals identified in the strategic level, there exists at least one node in the tactical level to execute this subgoal. This level is by nature data-rich, and therefore supports the use of a data-driven, forward chaining expert systems language such as Lisp or CLIPS.[2] The strategic level would only expect a boolean

value TRUE when each subgoal is complete, however, the data structures that reflect the results of each rule-based system have to be created and stored to pass to the ModSAF Orders Generator.

E. ModSAF ORDERS GENERATOR

The ModSAF Orders Generator converts the results from the Mission Selector/Evaluator into a set of ModSAF task frame stacks that are assigned to the company's elements (unit and/or vehicle). Under certain conditions, a company-level task frame could be generated that would affect all elements of that company. This module, and the ModSAF augmented, asynchronous finite state machines generated from this module comprise the execution layer of the Rational Behavior Model.

F. SUMMARY

The mission planner architecture consists of four modules, two of which are data conversion modules. The user is presented with a graphical user interface with a modified US Army operations order containing a limited set of input choices. The information from the operations order is converted into a form readable by the Mission Selector/Evaluator. The Mission Selector/Evaluator performs a "reasoning process," using the top two levels of the Rational Behavior Model as the framework about which the mission planning is performed.

The results of this reasoning process are presented to the user, who has the power to accept or reject the results. If accepted, the information is converted to a set of ModSAF tasks, which represents the lowest level of the Rational Behavior Model. If rejected, the user is presented with the operations order, where changes to the order can be made as required. This architecture can be implemented in various ways. Chapter V discusses the design strategies that were considered in the implementation of this architecture.

---

2. CLIPS: "C Language Integrated Production System" [GIARR93].

V. MISSION PLANNER DESIGN STRATEGY

Two courses of action were considered in the design of the mission planner. The first was to develop a stand-alone application that would interface with ModSAF through the Persistent Object Protocols -- the distributed strategy. The second course of action was to incorporate a library into the existing ModSAF code -- the integrated strategy. Both strategies share common design considerations, and have corresponding advantages and disadvantages. Both strategies were implemented to some degree, and the implications of each will be discussed in Chapter VI.

A. COMMON DESIGN CONSIDERATIONS

Certain aspects of both design strategies are common to both. The operations order graphical user interface (GUI) design, the extensive reuse of the existing ModSAF code, the design and incorporation of fuzzy sets for the expert systems, and the reasoning model are all common to both design strategies.

1. Operations Order Graphical User Interface

The OPORD GUI design strategy was to take advantage of the ability of ModSAF to create editors through their definition in text files by using the LibEditor library [SMITHc93]. This library permits rapid development and modification of GUIs in the OSF/Motif environment.

a. Operations Order Base Editor

The initial design of the base editor was to retain the basic ModSAF editor "look and feel" while at the same time presenting the user with a readily understandable format using minimal screen space (Figure 7). The base editor contains four separate sections: Assignment, Task Organization, Other Paragraphs, and "Paragraph 3 -- Execution." Since the focus of the OPORD is on Paragraph Three, a form of this paragraph is included in the base editor. A portion of the editor is dedicated to the display of the unit organization. This closely reflects the OPORD "Task Organization" portion, which is actually in Para-

graph One (Situation), but bears displaying in the main editor. The other two sections contain pushbuttons that allow the user to select the other editors in the set, or to assign, print, or cancel (exit) the operations order and return.

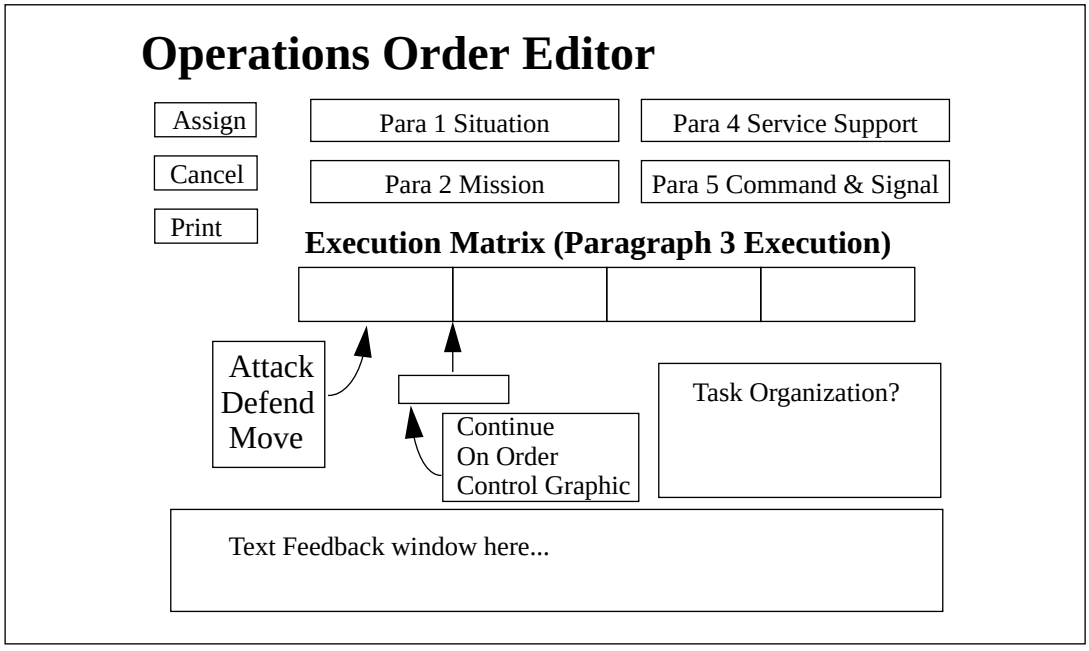


Figure 7: Preliminary Design -- Base Operations Order Editor

b. Operations Order Subordinate Editors

The conceptual design of the subordinate editors follows that of the top layer editor, but without the complexity (Figure 8). Paragraph One contains the essential graphical elements of the Situation Paragraph. The original intent behind the "Enemy Situation" and "Friendly Situation" pushbuttons was to provide a capability to graphically portray friendly and enemy elements on the tactical map. Paragraph Two was also graphically-oriented, depending on the ModSAF Overlay Creation editor set for data input.

Paragraphs Four and Five were developed to complete the OPORD process, and allow for future expansion to the considerations of supply status and chain of command selection in mission planning.

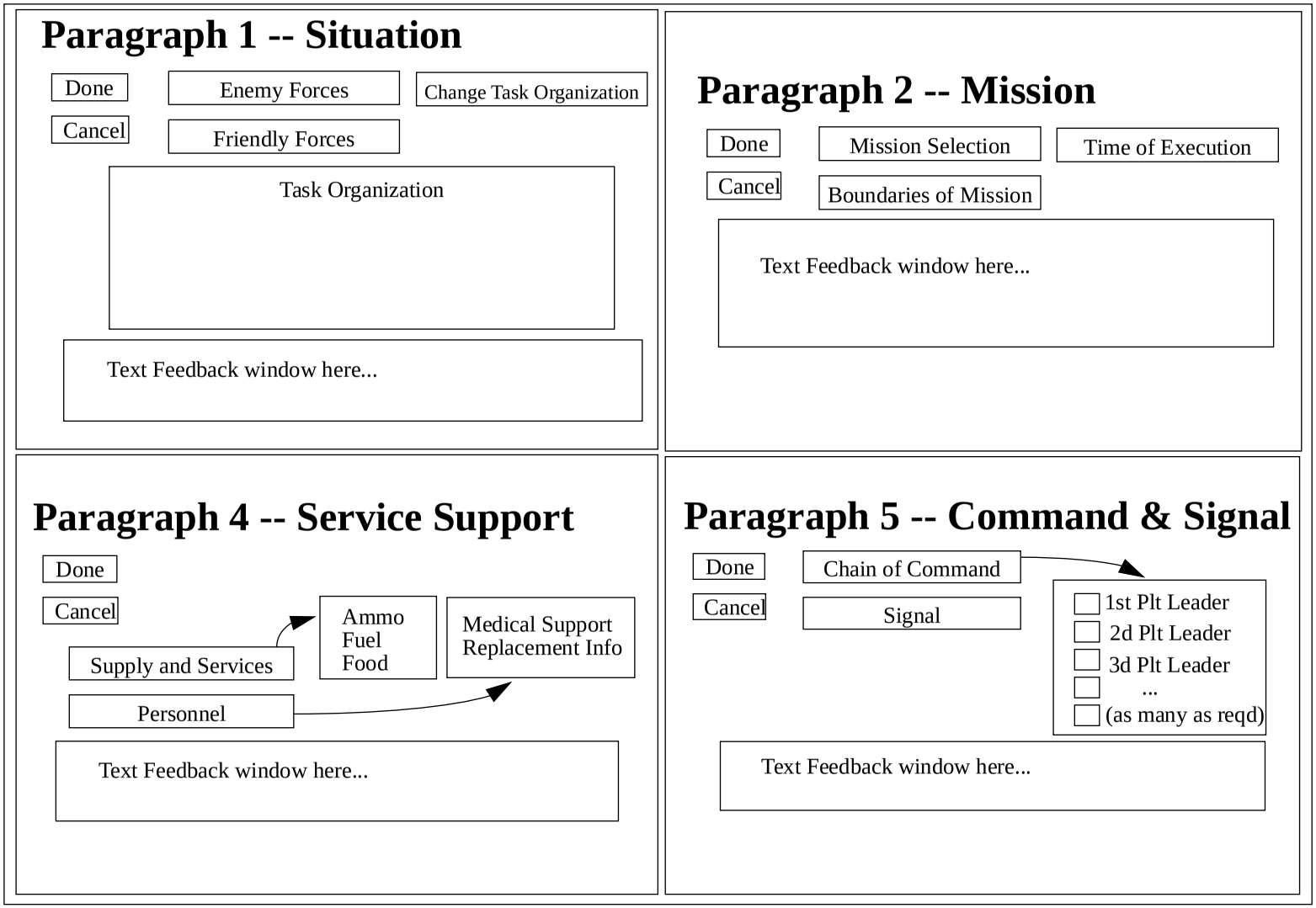


Figure 8: Operations Order Subeditors -- Paragraphs One, Two, Four, and Five

2. Fuzzy Set Design

Fuzzy logic and the use of fuzzy sets in the development of rule-based expert systems is a well-documented method for reasoning under uncertainty. Fuzzy set theory was first proposed by Zadeh in 1965 [ZADEH65]. Fuzzy logic involves testing for membership in these tests, and was proposed as a means for expert system reasoning under conditions of uncertainty [ZADEH79]. Today, fuzzy logic is an extensively used development tool for expert and embedded systems.

In the Mission Planner, fuzzy logic is to be used in the Tactical Level of the Rational Behavior Model to represent certain variables, and to determine a relative probability of mission success. This more approximates the level of uncertainty in battle, especially with regard to enemy situation, terrain, the actual meaning of the mission, the effectiveness of friendly forces, and the time available to plan prior to execution of the mission. One advantage of fuzzy systems is that the variables and their modifiers do not initially require actual fuzzy centroids to be implemented as rules. One can develop the fuzzy rules and test the overall system by assigning a "crisp" value to the set as a whole. While this may lead to inaccuracies in the initial prototype, this method allows rapid creation of the fuzzy rule set and the description of a particular environmental state using terms understandable to humans. Once the rules have been developed, then subsequent prototypes will include the fuzzy set operations and calculation of set membership using Max-Min Inference or Max-Product Inference and calculation of fuzzy centroids [DURK94].

The development of fuzzy membership sets into a rule-based expert system is described in [DURK94], and is broken down into a series of discrete tasks, as shown in Table 5. Tasks one through three include design considerations, and are enumerated below.

a. Task 1: Problem Definition

The Mission Planner must operate on a very large body of knowledge, which is best subdivided into smaller categories. The use of the Rational Behavior Model allows for compartmentalization of this knowledge, making each subcomponent smaller

and more efficient. Additionally, this strategy allows incremental development and testing, and provides an easy mechanism for expansion.

Task #	Action
1	Problem Definition
2	Define Linguistic Variables
3	Define Fuzzy Sets
4	Define Fuzzy Rules
5	Build System
6	Test System
7	Tune System

TABLE 5: Tasks Required to Build a Fuzzy Expert System, After [DURK94]

b. Task Two: Define Linguistic Variables

Linguistic variables (also known as fuzzy variables) describe the collected body of fuzzy knowledge. These variables will be used in the construction of fuzzy rules. All linguistic variables have a range of possible values, which are also defined in terms of their minima and maxima. The Mission Planner's set of linguistic variables is enumerated in Table 6 and graphically depicted in Figure 9. This set contains the essential elements of mission planning, derived from the US Army's acronym "METT/T"[1] [ARMY88]. They are:

- Enemy Forces: This is a force ratio compared to your own forces. 10:1 means there are ten friendlies to one known enemy; 1:10 means that the enemy is significantly stronger than you are.
- Intelligence Accuracy: Measures how accurate the intelligence picture is based on ground truth. This reflects the level of uncertainty about the known enemy forces -- 100% means totally accurate knowledge about the enemy strengths and dispositions and is related to the numbers and types of intelligence collection assets (such as scouts, etc.).
- Troops Available: At company level, the commander considers the total number of

1. Mission, Enemy forces, Troops available, Terrain, Time available.

:

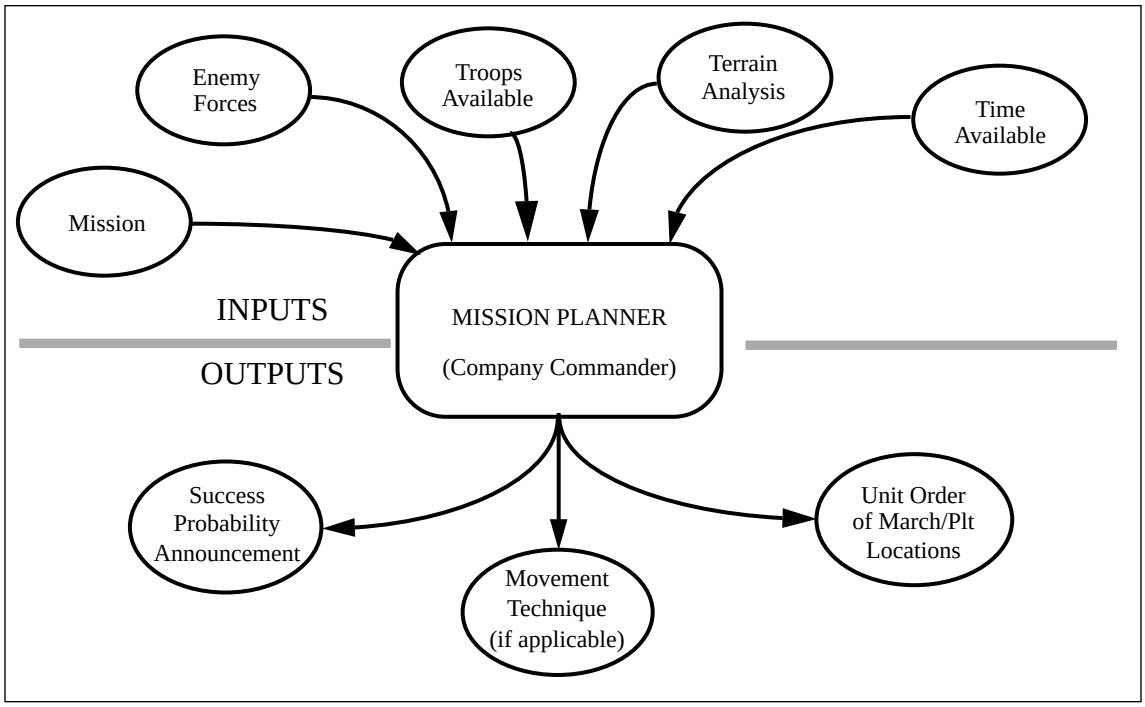


Figure 9: Inputs and Outputs of Mission Planner

Variable Name	Range	
	Min	**Max**
Enemy Forces	< 1:10	>10:1
Intelligence Accuracy	0%	100%
Troops Available	6 vehicles	24 vehicles
Time Available	1 hr	>24 hrs
Terrain Slope	0 degrees	30 degrees
Terrain Soil Type	> 4	1
Terrain Vegetation	0%	100%
Terrain Trafficability	0%	100%
Terrain Obstacles	0	>5
Success Prediction	0%	100%

Table 6: Linguistic Variables and Their Ranges

combat vehicles (tanks and infantry fighting vehicles) available to him for a particular operation. If the company has six vehicles, then it is at less than 50% strength and is considered to be barely combat effective. The most a commander can effectively control is five platoons for a total of 24 vehicles.

- Time Available: The total amount of time available to the commander to plan and prepare for the operation. The more time available, then the more opportunities to rehearse, receive reinforcements, and to prepare positions.
- Terrain Slope: Measured in degrees -- the steeper the slope, the more difficult the route (hence slower speeds required). If the slope is greater than 30 degrees, then this is considered impassable.
- Terrain Soil Type: A hard-surface road (gravel or improved) has a value of 1; any value greater than 4 is impassable. This could be expanded to include all 16 soil types defined in the compact terrain database library, but for now will be limited to these four [SMITHb93].
- Terrain Vegetation: 100% vegetation hinders movement but maximizes concealment from the enemy.
- Terrain Trafficability: A combination of the slope, soil type, and vegetation.
- Terrain Obstacles: These are considered to be man-made versus natural obstacles (natural obstacles are covered in Terrain Trafficability). If there are greater than five obstacles, then the route could be considered to be impassable without significant engineering support.
- Success Prediction: This is a fuzzy value derived from combining the enemy force ratios, intelligence accuracy, terrain trafficability, and terrain obstacles to derive a company commander's prediction of success.
- Movement Technique: This is applicable in Attack and Move missions. In the attack, there are three basic movement techniques -- travelling, travelling overwatch, and bounding overwatch. In a movement, there are essentially two -- road march and travelling.

c. Task Three: Define Fuzzy Sets

The fuzzy sets derived from the linguistic variables are now defined in terms of modifying adjectives. Table 7 enumerates a list of adjectives that will be used with each linguistic variable.

Fuzzy sets are now constructed using the modifying adjectives. These are defined in Appendix B, and include fit vectors. The use of fit vectors ensures that all fuzzy sets have sufficient overlap to assure that every possible value establishes some fuzzy set membership value.

Enemy Forces	Intelligence Accuracy	Troops Available	Time Available	Terrain Slope
Impotent	Erroneous	Ineffective	None	Level
Weak	Inaccurate	Weak	Short	Gentle
Parity	Questionable	Company Minus	Moderate	Moderate
Strong	Accurate	Normal	Long	Steep
Overpowering	Reliable	Reinforced	Extended	Precipitous

Terrain Soil Type	Terrain Vegetation	Terrain Trafficability	Terrain Obstacles	Success Prediction
Improved	Open	Impassable	Zero	Zero
Normal	Thin	Difficult	Light	Problems
Difficult	Moderate	Moderate	Moderate	Maybe
Impassable	Thick	Easy	Dense	Good
	Dense	Smooth	Impassable	Outstanding

Table 7: Linguistic Variables and their Modifying Adjectives

3. Reasoning Models -- Expert System Submodules

At least four reasoning models must be constructed in the Tactical level that employ fuzzy logic. These are: the terrain reasoning submodule, the attack submodule, the defend submodule, and the move submodule. Additional models may be constructed as needed, to complete the inclusion of all aspects of METT/T; however, some may be best represented empirically through the user interface.

a. Terrain Reasoning Submodule

The terrain reasoning submodule is a general-purpose model that provides input to the others through the selection of movement routes for the Attack and Move submodule, and identification of possible enemy routes of advance in the Defend submodule. It will use the Compact Terrain Database as its input, and will require some additional preprocessing to "fuzzify" the data.

b. Attack Submodule

This submodule contains the elements required for a company to plan an attack. Planning elements include objective identification, route determination, (using the Terrain Reasoning submodule), enemy forces throughout the area of operations, and any restrictions that may be imposed on the company by the battalion commander, such as following an axis of advance, and boundaries. Additionally, this submodule needs to determine the nature of the attack and select the attack type appropriate to the mission, which has a bearing on the final disposition of the company after the attack. There are two types of attack: terrain-oriented and force-oriented.

A terrain-oriented attack is rarely conducted unless absolutely required for the success of a particular mission. Such terrain is considered to be *critical terrain*, and is required to be physically occupied. An example of this is a single bridge crossing an unfordable river. In this case, all considerations of enemy force destruction are secondary to the attaining of the physical objective.

A force-oriented attack is the more common type. In this form of attack, the objective is the most probable location of the enemy's force that must be destroyed. This means that commanders can adjust the location of the objective based on the updated enemy situation, as long as the goal is achieved -- destruction of the enemy force.

c. Defend Submodule

Since the defense naturally assumes that the enemy force is stronger (or at least, has the initiative), the decisions to be made here are less concerned with the size of the enemy force; focusing instead on the ability of the company to effectively engage at the maximum range of their weapons systems, and ensure mutual supporting fires. Planning considerations for the defense include available preparation time, the orientation and direction of the enemy's main attack, and determination of possible enemy avenues of approach. Like the attack submodule, there are different considerations if the defense is terrain-oriented or force-oriented.

d. Move Submodule

A company-level maneuver force will employ two basic forms of movement: tactical and administrative. Tactical movement is incorporated into the above two submodules as a means of attaining the overall goal. This submodule reasons about administrative moves, or road marches.

A move mission is performed when a company must travel from one location to another, usually within its own territory. As a result, the likelihood of enemy contact is much less than with the previous two missions. However, this does not mean that all considerations of enemy capabilities can be discarded. Ground maneuver companies in a road march must be constantly on the lookout for enemy fixed wing and rotary wing aircraft, partisan activities, and enemy deep strikes. Also, if the known enemy situation is not clear, then additional security precautions must be made.

B. DISTRIBUTED APPLICATION STRATEGY

This strategy involves building a separate application that incorporates ModSAF libraries but performs no simulation (Figure 11). It would communicate with the ModSAF stations through the Persistent Object (PO) Protocols, and have access to the PO Database. The user would see the same Plan View Display as the ModSAF station, less some tool icons. The option to view individual vehicles would also be optional, as the user would not directly control the company from this station.

The user would have the ability to create the unit, select it, and generate an operations order using the GUI described previously. The application would require a configuration of ModSAF to be active and communicating on the network using the same PO Database and exercise. Additionally, the station would be required to read DIS packets to monitor the situation.

The advantages of this strategy include code reuse and simplification, since this is essentially a ModSAF application less the capability to generate and modify task frames and execute augmented asynchronous finite state machines. This would reduce the taxation of system resources, allowing room for the artificial intelligence modules to be inserted. Additionally, this stand-alone application could be written in a truly object-oriented language, such as C++ or Ada 9x, thus providing a means to encapsulate data for use by object-oriented artificial intelligence languages such as Common LISP or the CLIPS Object-Oriented Language (COOL).

The disadvantages of this strategy include the requirement to update the code every time a new version of ModSAF is generated. Additionally, many of the libraries are tightly coupled to each other; selection of which libraries to include and, more importantly, which ones to reject require a significant investment in time and resources.

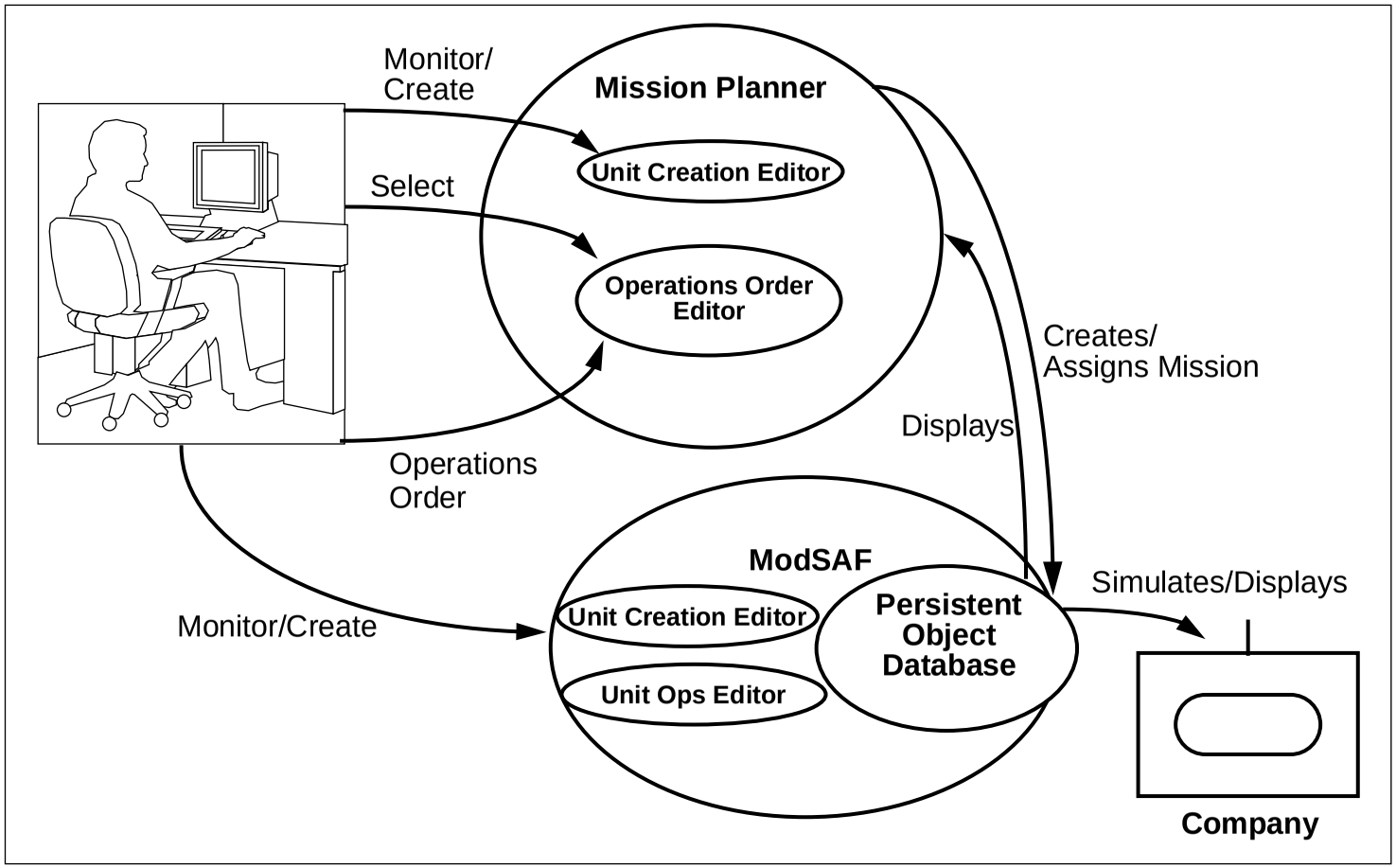


Figure 10: Distributed Design -- Mission Planner Separate From ModSAF

C. INTEGRATED APPLICATION STRATEGY

This strategy involves using ModSAF as the basic application, and simply adding a separate library to the system (Figure 11). The library would contain the operations order GUI, along with the ability to call compiled artificial intelligence submodules.

The user, through the ModSAF application, creates the company-sized unit from the Unit Creation editor. The user then selects that unit by clicking on its icon, or one of the vehicles from the tactical map, which brings up the Unit Operations Editor. The "Operations Order" button can now be pressed to launch the to start the operations order.

The advantages of this strategy include simplicity and economy of workstation resources. A separate workstation is no longer required, and incremental testing can be performed quickly, as no network resources are required (one could test the library from within a combination SAFsim and SAFStation disabling the network protocols). Additionally, this strategy follows closely the intent behind ModSAF's acronym -- modularity. It simply adds one more library to the hundreds already present.

The disadvantages of this strategy include increasing the complexity of an already complex application, and the lack of system resources such as random access memory and CPU cycles to execute the artificial intelligence submodules. A possible solution is to allow the OPORD to be selected only if the workstation is a SAFStation, and the simulation augmented, asynchronous finite state machines are executed by a separate SAFsim.

D. SUMMARY

Two courses of action were identified that shape the design of the mission planner. One course of action is to develop a stand-alone system that uses ModSAF's distributed architecture to communicate with the SAFStation and SAFsim. The second course of action is to develop a library (module) within the ModSAF architecture itself. The distributed strategy allows for a greater degree of freedom in the selection of the programming language, and the integration of artificial intelligence languages within the system. The integrated strategy is simpler to implement, and requires less modification of existing ModSAF

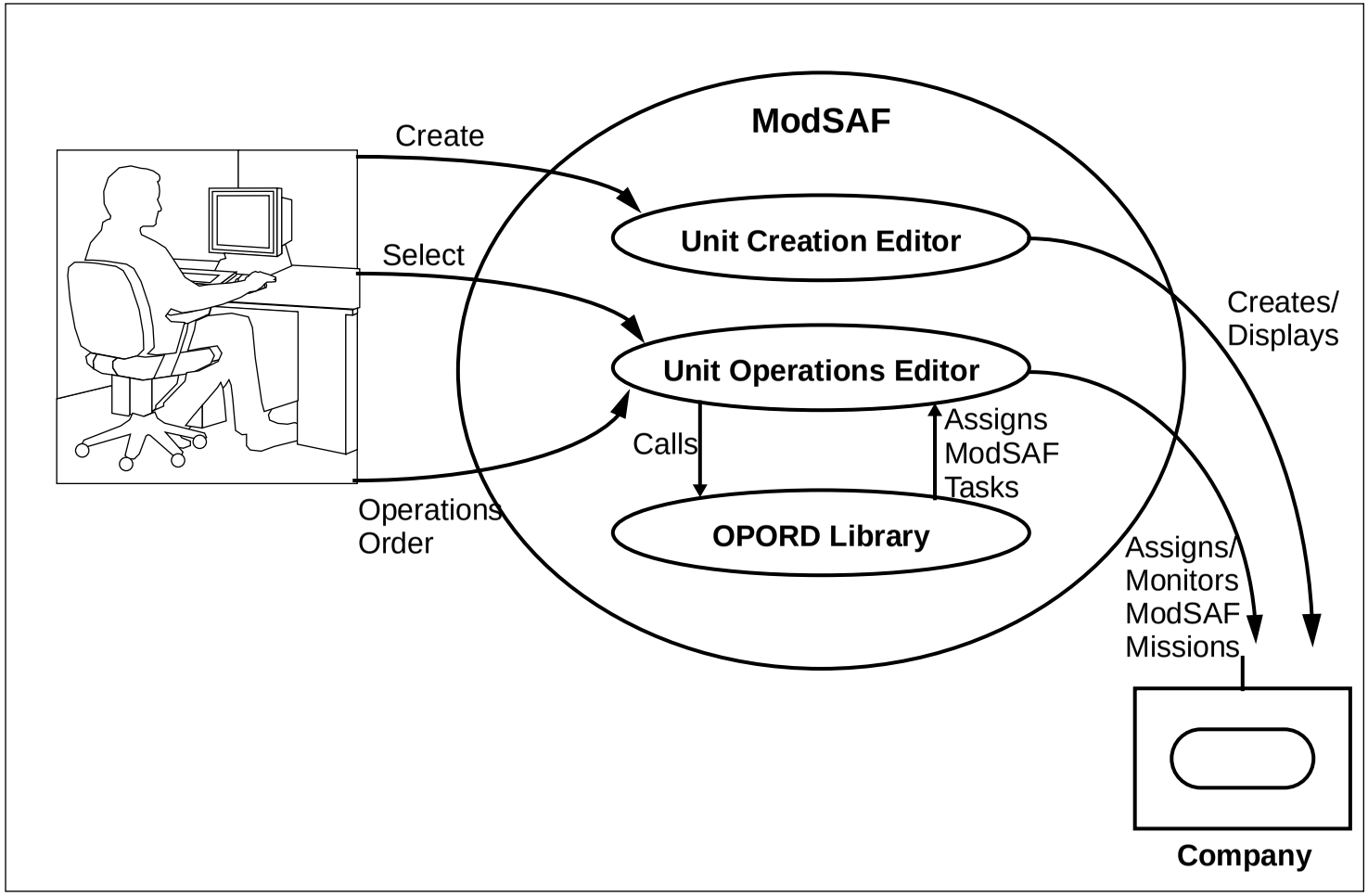


Figure 11: Integrated Design -- Mission Planner Part of ModSAF

code. The common factors in the design of the Mission Planner include the development of an intuitive graphical user interface for data input, the selection of a reasoning strategy, and identification of the initial reasoning submodules. The next chapter discusses the development issues that arose in pursuing these courses of action, along with the solutions, if any, that were found.

VI. MISSION PLANNER DEVELOPMENT

The development of the mission planner proceeded initially by attempting the implementation of the distributed design strategy. The methods used to implement this strategy were found to have significant shortcomings. Fixing these problems would have required a major revision of the existing code. This was determined to be a much more significant effort than the remaining research time allowed, so the integrated strategy was implemented using a minimalist approach. This was successful.

The distributed strategy is still viable, but requires a different implementation method and additional resources to fully develop it into a working prototype. This chapter discusses both implementations, and concludes with recommendations for further development.

A. INITIAL ATTEMPT: DISTRIBUTED DESIGN STRATEGY

The initial effort was focused on the development of a stand-alone application that would allow the use of a truly object-oriented language, such as C++. This language, unlike K&R C, supports stronger type-checking, polymorphism, and object inheritance. This was initially selected to make the application development easier, as C++ supports stronger type checking, and the use of objects for mission selection would closely follow the object-oriented paradigms in the Common Lisp Object System (CLOS) and CLIPS [GIARR93] [STEE90][STRO91].

ModSAF libraries were used exclusively to share the PO database, define network protocols, and create the plan view display. This required a rewrite of the main.c code to allow it to be compiled in C++, and the dual definitions in all public header files of the function prototypes, to support both K&R C and C++. Table 8 summarizes the changes required for each affected library. This prototype application was basically a collection of 67 ModSAF libraries with a C++ wrapper, and was to be the basis for the development of the mission planner. Unfortunately, this effort was an extremely tedious and time-consuming, as a total of 47 libraries required modification of function prototypes. The ModSAF libraries at the lower layers of execution are tightly coupled to each other, and the time spent at-

Item	Library	File Name	Function Name(s)
1	libbgr	libbgr.h	typedef struct BgrDC[a]
2	libassoc	assoc.h	AssocTickAssocLayer (int32)
3	libassoc	assoc.h	added #include <stdtypes.h> for int32
4	libpo	libpo.h	All public functions
5	libqueue	libqueue.h	All public functions
6	libpktvalve	libpktvalve.h	All public functions[b]
7	libreader	libreader.h	All public functions[c]
8	libp2p	p2p.h	All public functions
9	libcallback	libcallback.h	All public functions[d]
10	libtime	libtime.h	time_init()
11	libsched	libsched.h	All public functions
12	libcmdline	libcmdline.h	All public functions
13	librdrconst	librdrconst.h	All public functions
14	libotmatch	libotmatch.h	All public functions
15	libechelondb	libechelondb.h	All public functions
16	libformationdb	libformationdb.h	All public functions
17	libdisconst	libdisconst.h	disconst_init()
18	libphysdb	libphysdb.h	physdb_init()
19	libpo	libpo.h	all handler types redefined
20	libparmgr	libparmgr.h	all public functions
21	libvtab	libvtab.h	vtab_init(), vtab_create_list().
22	libpbtab	libpbtablh	pbt_init(), pbt_set_size()
23	libentity	libentity.h	ent_init(), ent_set_minimum_pbt_error()
24	libstealth	libstealth.h	stealth_init()
25	libc2obj	libc2obj.h	c2obj_init()
26	libquad	libquad.h	All public functions
27	libctdb	libctdb.h	All public functions

TABLE 8: Library Modifications to ModSAF for C++ Implementation

Item	Library	File Name	Function Name(s)
28	libcoordinates	libcoordinates.h	All public functions
29	libroutemap	libroutemap.h	All public functions
30	libtactmap	libtactmap.h	All public functions
31	libsafgui	libsafgui.h	All public functions
32	libprivilege	libprivilege.h	All public functions
33	libpvd	libpvd.h	All public functions
34	libeditor	libeditor.h	All public functions
35	libxfile	libxfile.h	All public functions
36	libsensitive	libsensitive.h	All public functions
37	libbgrdb	libbgrdb.h	bgrdb_init()
38	libbgr	libbgr.h	BgrInitWithDisplay()[e]
39	libselect	libselect.h	All public functions
40	libview	libview.h	vw_create()
41	libgraphics	libgraphics.h	All public functions
42	libunits	libunits.h	units_create_editor()
43	libopord	libopord.h	convert to c++
44	libdelobj	libdelobj.h	All public functions
45	libtdbtool	libtdbtool.h	All public functions
46	liboverlay	liboverlay.h	All public functions
47	liblocalmap	liblocalmap.h	localmap_init(), localmap_set_tactmap()

TABLE 8: Library Modifications to ModSAF for C++ Implementation

a. In the file libbgr.h (in libbgr library), the following was changed: typedef struct BgrDC { to:

```
#if defined(__cplusplus) || defined(c_plusplus)
   typedef struct {
#else
   typedef struct BgrDC {
#endif
```

b. Included libshmif.h to libpktvalve.h, and stdtypes.h to libentity.h.

c. Changed the variable declaration in reader_set_search_paths from "default" (reserved word) to "defalt" (this is what was declared in rdr_parser.c and rdr_parser.y).

d. Also required an explicit type cast to type CALLBACK_HANDLER from type int32.

e. This function was declared without any references to the required arguments so the elipsis (...) was used.

tempting to determine their dependencies was significant. Complicating this effort was the fact that certain variables were C++ reserved words. For example, all the libraries had to be modified to remove instances of “class” and replace them with “mclass,” and “new” with “mnew.”

Additionally, all the function prototypes in Table 8 required two separate definitions -- one for K&R C and the other for C++. Each of the arguments to the functions had to be declared, or at least defined with an ellipsis (...). This alerted the C++ compiler that the arguments would be defined at a later time within the compilation process. The following is an example without the ellipsis, from the Compact Terrain Database library (libctdb):

```
/* ctdb_point_on_database returns 1 if the point is on the database,
 * 0 if it is not.  All libctdb functions make this check internally.
 */
#if defined(__cplusplus) || defined(c_plusplus)
/* Defines a C++ function header */
       int32 ctdb_point_on_database(  CTDB *ctdb,
                                      float64 x,
                                      float64 y  );
#else /* Not defined c_plusplus */
       extern int32 ctdb_point_on_database( /* CTDB *ctdb,
                                               float64 x,
                                               float64 y  */ );
#endif
```

Fortunately, the programming style guide required the definition of the arguments within comments, so the majority of the functions were relatively easy to convert. The difficulty arose when an argument was of a different type than was defined. In K&R C, type definitions are not as stringently monitored as in C++. Thus, the argument either had to be cast to the appropriate type, or the ellipsis used, as shown below, from libbgr.h. This particular function was an excellent example of writing obtuse and unreadable code. However, C++ allows the ellipsis, and the function was redeclared:

```
#if defined(__cplusplus) || defined(c_plusplus)
       int BgrInitWithDisplay ( ... );
#else /* Not defined c_plusplus */
       extern int BgrInitWithDisplay ();
#endif
```

The end result was an application in which 66 out of 67 libraries were written in K&R C (virtually unmodified ModSAF source code), one library was written in C++, and the

main.C file was written in C++. The application allowed the creation of a unit, and the display of platoon-level units and larger. Smaller units and individual vehicles were not represented.

This application, however, was somewhat unstable. The ModSAF system had to be running with no problems, such as excessive state transitions, or it would crash. The debugging effort was tedious and long, due to the requirement of maintaining at least a SAFsim on the network throughout. Libraries were included that were never used but were included by libraries that were used. This needlessly inflated the code and introduced the possibility of undesirable side effects occurring.

B. RE-EVALUATION OF DEVELOPMENT APPROACH

The release of ModSAF version 1.2 forced a reevaluation of the development process. This release incorporated major changes in the code, making Version 1.0 and Version 1.2 incompatible with each other. This resulted in a complete change of approach, as the redefinition of function prototypes would have to be repeated for all libraries in ModSAF 1.2 that were used by the mission planner. The use of C++ in the application development was becoming more difficult to implement than it was worth.

A number of lessons were learned in pursuing this approach. The selection of a particular language depends significantly on the language of the preexisting code. Unless one is willing to do a complete rewrite of the program, the programming language should be the same as the majority of the code that will be reused in the new application. A minimalist approach to code modification and extensibility should be pursued whenever possible. In attempting to use the large body of preexisting ModSAF code, changes were made that were inconsistent with good programming practices. The integration of the new code with the old code was not well-defined, and thus allowed inconsistencies in the application's execution.

C. SECOND ATTEMPT: INTEGRATED DESIGN STRATEGY

The second attempt was much more successful, and involved the building of a separate library and incorporating it into the existing ModSAF code. This new library -- "LibOpord" -- was created and integrated in the same manner as the other subordinate ModSAF libraries. Analysis of the code structure for both versions of ModSAF revealed that the unit operations editor was the best choice for the insertion of the code to initialize and call LibOpord. This is a base editor defined in the LibUnits library that allows the user to enter a set of tasks for the selected unit, its subordinate units (if any), and individual vehicles [SMITHe93]. It links these tasks together through operator defined phase transitions, called enabling tasks [SMITHd93]. The unit operations editor was chosen as it is the one that appears when a unit or a vehicle is selected from the Plan View Display. As a result, a minimal change to one existing ModSAF library was required, in addition to the inclusion of the header file in the main.c preprocessor directives.

1. Integration of the Mission Planner Into ModSAF

The integration of LibOpord into the ModSAF library set was done in accordance with [LORAa93]. The library requires the following modifications to LibUnits to become available to the user:

- Modification of the data structure within LibUnits to include a Motif pushbutton widget that will call the LibOpord editor, add a pointer to the LibOpord data structure, and define LibOpord as an additional sub-editor.
- Inclusion of the initialization function within the LibUnits initialization routine ("units_create_editor(...)) that will allocate memory for the data structures and build the Motif GUI widget tree.
- Addition of a callback (units_operations_order(...)) within LibUnits that handles the pushbutton mouse event.

Initialization of LibOpord is done as part of the LibUnits initialization steps; no other library requires modification. This form of library initialization and utilization is identical to the way other ModSAF editors are created and called.

2. Graphical User Interface Development

The base operations order editor was intended to be built using the LibEditor functions, but this proved unfeasible due to the irregular nature and complexity of the editor. Instead, the editor was created using a Motif widget tree that allows the programmer to build a customized GUI (Figure 12). The root of the widget tree is attached to the ModSAF base GUI, forming a branch that is displayed when called [SMITHf93].

The subordinate editors were developed using the LibEditor library [SMITHc93]. The LibEditor create function is called in the LibOpord initialization function for each subordinate editor. Currently, there are nine subordinate editors that are initialized in this manner. Every subordinate editor has two corresponding functions that are called during runtime when the editor is displayed. The first function hides the base editor and calls a LibEditor function to display the selected editor. The second function collects the user input data when the editor is exited and control returns to the base editor.

3. Implementation Limitations

The OPORD editors constrain the user to a limited set of choices, which is significantly different than a free-text OPORD. There are several obvious reasons for this:

a. Natural Language Processing Limitations

The limitations inherent in natural language processing do not allow for rapid integration of the data input to the other modules in the mission planner. This problem is a subject of ongoing research; such a data entry system would be too complex and cumbersome to implement here.

b. Mission Simplification

One of the goals of the mission planner is to simplify mission determination and selection by the human. An extremely rich OPORD editor would only serve to complicate the generation of company level missions. Instead, a robust expert system should

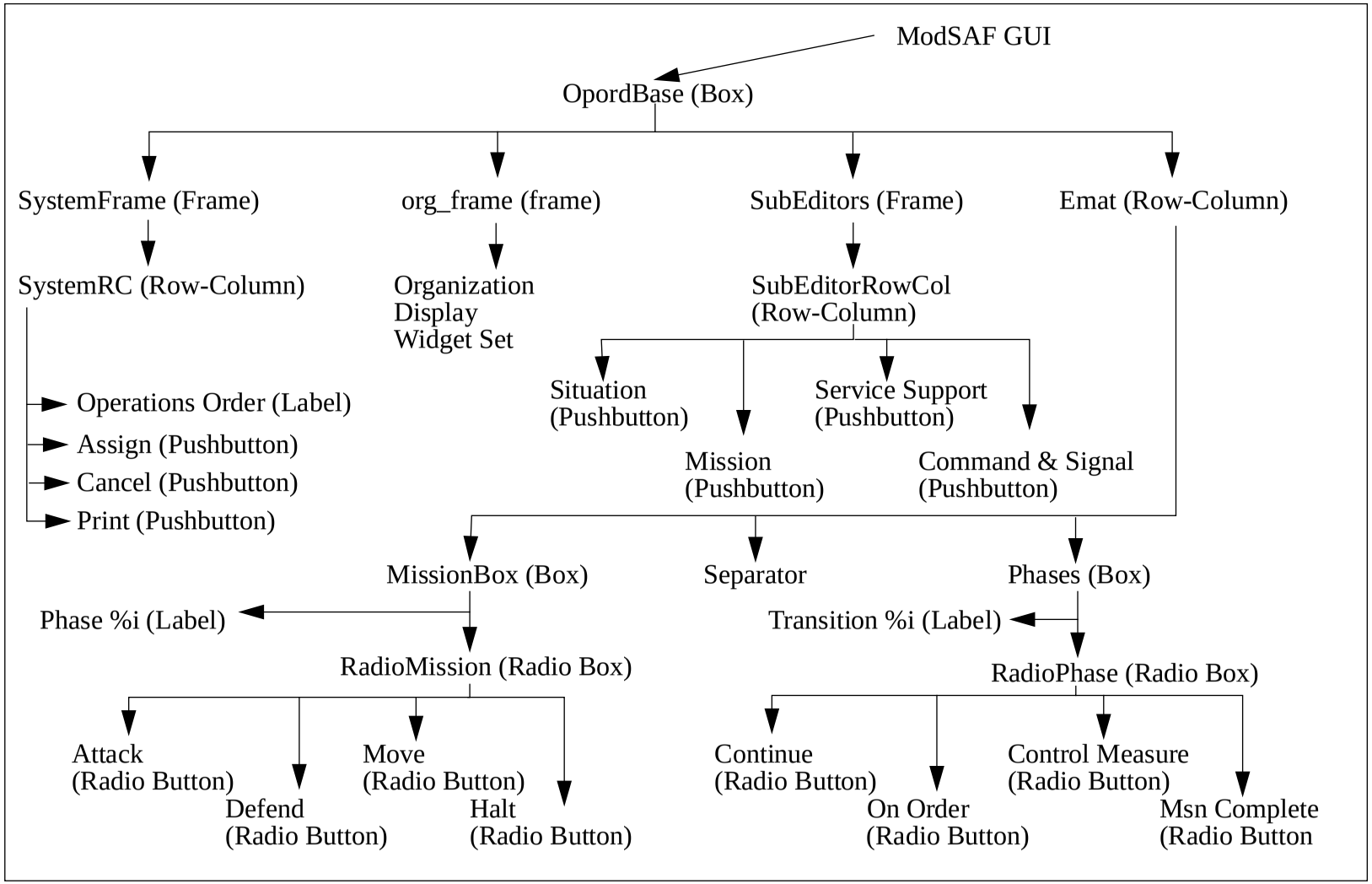


Figure 12: LibOpord GUI Widget Tree

be able to compensate for the simplicity of input by reasoning about the circumstances of the input data and making decisions in the context of the assigned mission.

4. Data Formatter

The purpose of the Data Formatter is to ensure the artificial intelligence submodules of the Mission Selector/Evaluator receive the user input data in a usable form. The Data Formatter is a set of data structures and the code that converts the data from one structure to the other. Its current implementation is as a structure of structures. There is currently no modification being done as the artificial intelligence submodules are not developed. The user interface through LibEditor requires that the data displayed and entered must be placed in a separate structure for later processing. To simplify this procedure, a base structure is defined that contains the five paragraphs in separate structures (Figure 13). This compartmentalization of data allows one portion of the structure to be modified based on the user's selection.

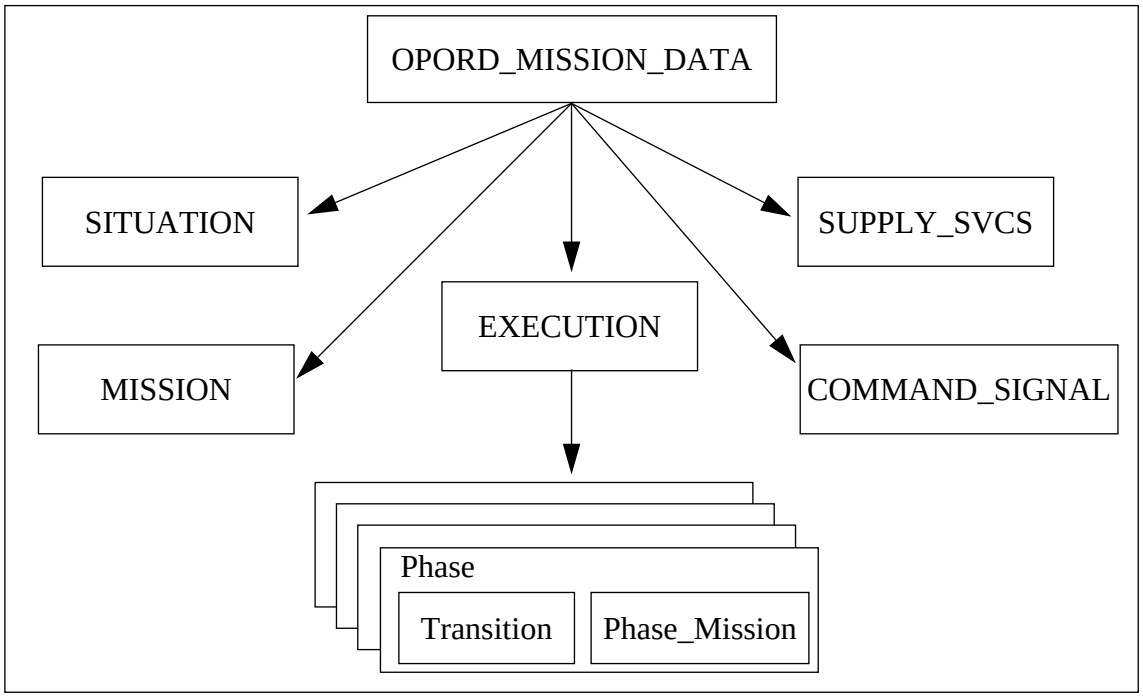


Figure 13: Compartmentalized Data Structure

The OPORD_MISSION_DATA structure aggregates the information from all editors into one structure. This approach provides the capability to easily change the data parameters in one centralized structure. This area is ripe for additional enhancements; the main danger here is overwhelming the user with data. The focus here is to request the essential data (keep it simple) and let the AI reason about the context and situation, and modify those parameters as required. Currently, the maximum number of phases for a given operations order is four. This limitation was a design choice. Most battalion-level operations orders never exceed four or five phases; this number can be changed through adjusting the value of the array variable through the constant MAX_OPORD_PHASES.

5. Mission Selector/Evaluator

This module was not implemented due to the time constraints involved. The shell about which the Mission Selector/Evaluator operates was successfully completed, but the initial attempts to develop a distributed mission planner consumed the available time. This submodule could be the subject of future work, as will be explained in Chapter VIII.

6. ModSAF Orders Generator

The ModSAF Orders Generator is also a prototype module. It makes extensive use of the new LibTaskUtil library, which is designed to allow direct creation of specified task frames without calling the task's associated editor[SMITHb94]. This allows libraries like LibOpord to generate a set of task frames and assign them to a unit, without human intervention. The library was modified by J. E. Smith to allow multiple task frames to be linked by enabling tasks. These modifications will become generally available in the next version update of ModSAF (Version 1.3). A single reader file was modified to include company-level tasks; the associated editors and libraries were passed to the taskutil_init function during initiation of the operations order structures and editors.

D. TERRAIN REASONER DEVELOPMENT

One of the artificial intelligence submodules is partially complete. The terrain reasoner is a mission-independent submodule that will reason about the terrain in its area of operations and determine whether a particular route is "Slow-Go," "No-Go," or "Go" terrain. Its current implementation is as a stand-alone module that accesses a text data base with the same basic structure as the Compact Terrain Data Base. The terrain analyzer was implemented using CLIPS Version 6.0, and requires approximately one minute execution time for a three-kilometer square area.[1]

The terrain analysis module will use the posts from the Compact Terrain Database as the map (with additional information as needed from the quadtree database) [STAN93], and the boundaries as specified by the overlay in the "Paragraph One: Situation" editor to generate a set of points. There is a one-to-one correspondence between the posts within the boundaries of the overlay and the points for terrain analysis. The points are uniquely identified by an integer number and by their location on the map using UTM or x-y coordinates.

Once the points have been defined, the process of determining mobility corridors between them begins. This process requires a user-defined mobility threshold to assist in the analysis of mobility corridors between the points. The default value is "smooth trafficability." The algorithm moves along each established point and analyzes the trafficability between the current point and each point adjacent to it. If the trafficability meets or is better than the established threshold, then a new mobility corridor is established between those two points. If a mobility corridor already exists, then the adjacent point is skipped, and processing continues with the next adjacent point. This process is continued for every established point. The end result is a network containing all points and their attached mobility corridors (Figure 14).

Following the creation of the mobility corridors, avenues of approach (routes) are then determined. Essentially, these avenues of approach attempt to move from a starting point

1. Using a Silicon Graphics Indigo XS, with 100MHz R4000 processor.

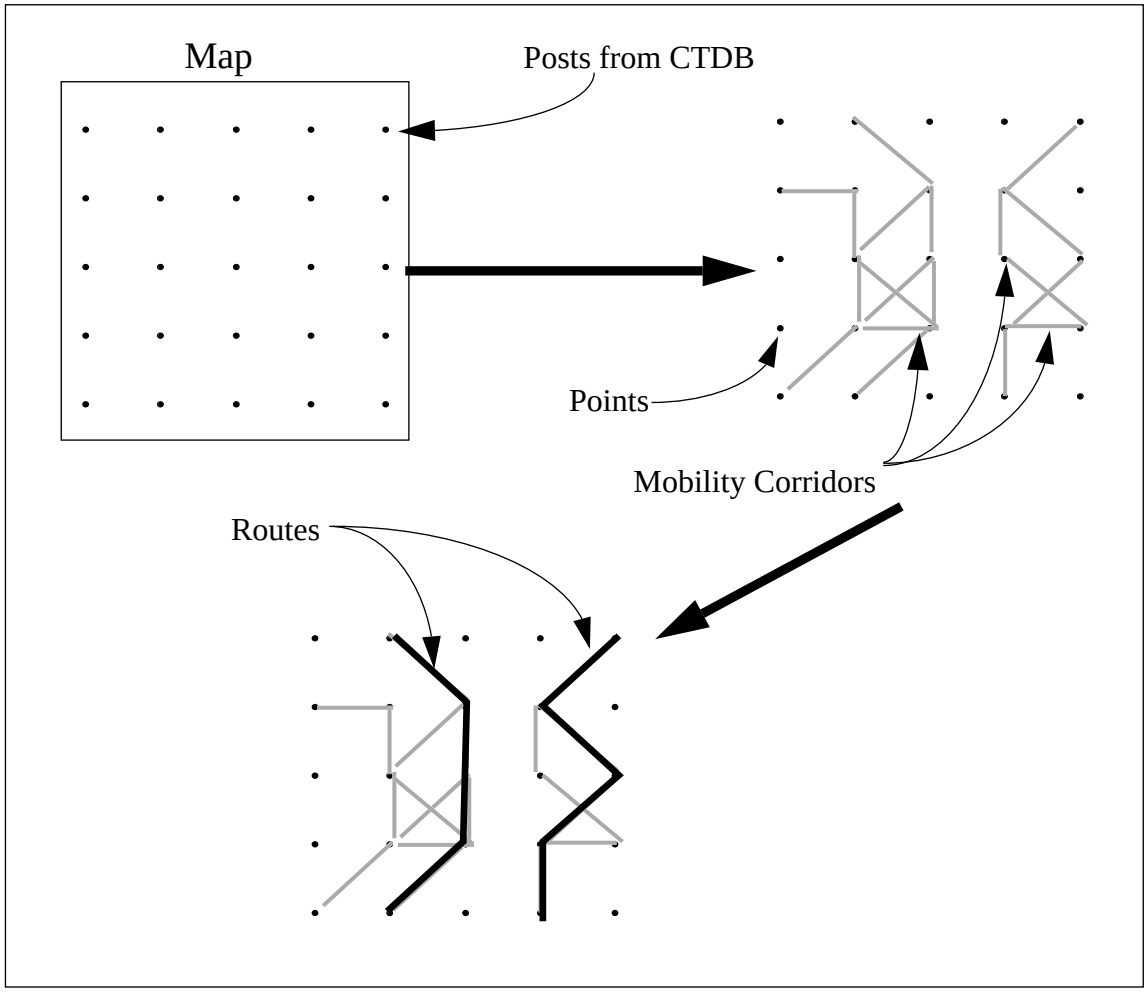


Figure 14: Terrain Analysis Steps

(usually the attack position in an attack) to the objective. Knowing the starting point and the ending point, the intermediate points (i.e. mobility corridors) are selected that offer the best trafficability (or, a route that offers a user-defined trafficability range). In its current implementation, an A-Star search is performed to determine the least-cost path from the start point to the goal point.

The A-Star algorithm used the following heuristic:

$$f(n) = g(n) + h(n), \text{ where} \quad \text{(Eq 6.1)}$$

$$g(n) = traffic(p, q) \times dist \quad \text{(Eq 6.2)}$$

$$traffic(p, q) = \begin{cases} 1, \text{ when } c((route) \geq c(mc)) \\ c(mc) - c(route) + 1, \text{ when } c(route) < c(mc) \end{cases} \quad \text{(Eq 6.3)}$$

$$c(x) = \begin{cases} 1, \text{ when trafficability is SMOOTH} \\ 2, \text{ when trafficability is EASY} \\ 3, \text{ when trafficability is MODERATE} \\ 4, \text{ when trafficability is DIFFICULT} \end{cases} \quad \text{(Eq 6.4)}$$

$$h(n) = \left(\sqrt{(currentx - goalx)^2 + (currenty - goaly)^2} \right) \times 125.0 \quad \text{(Eq 6.5)}$$

The POINT object is used to define the endpoints of the mobility corridors (See Figure 15). It includes all connecting mobility corridors that intersect at that point. Other slot values encapsulate information from the Compact Terrain Database.

The Mobility Corridor (MC) object is used to define a mobility corridor. It contains the two points that uniquely identify the mobility corridor's location, and the terrain attributes for that mobility corridor: slope, soil, vegetation, obstacles, overall trafficability, and distance. If the terrain is impassable between a set of two adjacent points then no MC is generated.

The ROUTE object is used to define a single route using a set of mobility corridors. It uses the MC and Point objects to define its location. The ROUTE object contains a list of

points that comprise the route. It also contains the overall trafficability of the route, and the distance.

Terrain analysis in this case is the consolidation of four fuzzy sets into a single fuzzy set. The input sets are Terrain Slope, Terrain Soil Type, Terrain Vegetation, and Terrain Obstacles. The output set is Terrain Trafficability. Essentially, the rule set attempts to determine terrain trafficability based on the state of membership of the other four sets.

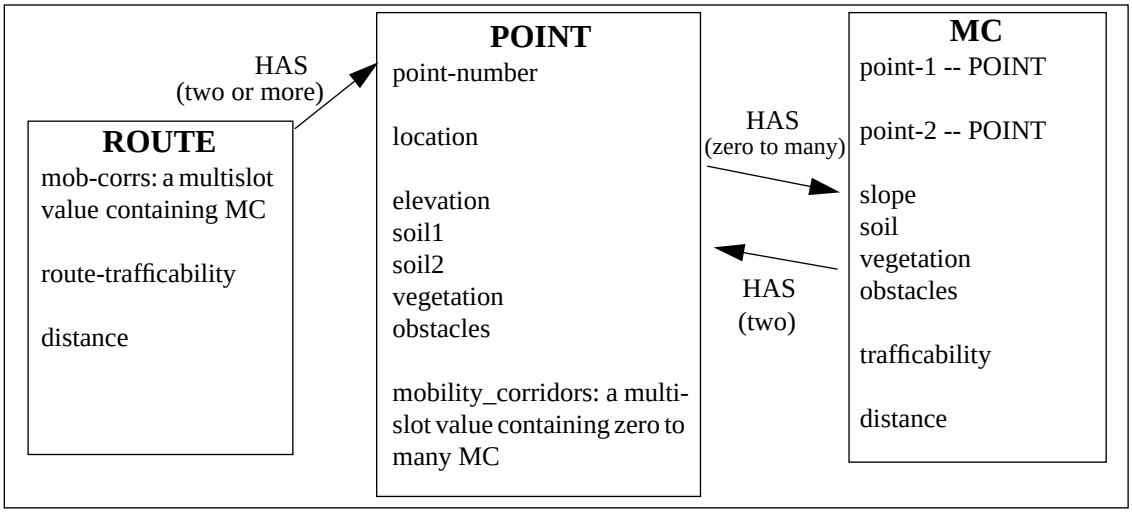


Figure 15: Terrain Analysis Object Hierarchy

E. SUMMARY

The distributed design strategy was more involved, but was probably due to faulty methodology than the strategy itself. The integrated design strategy resulted in the rapid development of a proof-of-concept prototype. While this strategy is the simpler of the two, it may be rendered unusable due to the potential resource requirements of the artificial intelligence modules. The prototype terrain reasoner, written in CLIPS, requires 60 seconds to determine a single route using A* search on a three kilometer by three kilometer terrain. Expanding this to cover an "average" battalion area of interest of five by five kilometers could easily triple the time required. Additionally, this would require heavy use of system resources, which may not be available due to the demands of ModSAF. Combining this submodule with other expert system submodules may make the integrated design strategy

unfeasible, however, its advantages are the ability to rapidly develop and test a module within the framework of ModSAF. In Chapter VII, the results of the development process are presented.

VII. RESULTS OF WORK

The results of the mission planner development are shown in the following diagrams. They depict the creation of an M1 tank company, the operations order editor screens, and the final results of the operations order process -- namely, a set of ModSAF task frames. The operator can select from four basic missions: Attack, Defend, Move (Road March), and Assembly Area. The transitions between phases can either be a timed duration (such as 45 Minutes from assignment of the order, or end of the previous task), a maneuver graphic control measure selected (or created) from the overlay, or to continue once the current task is complete.

A. SYSTEM PERFORMANCE

Since no actual reasoning process is occurring, there is virtually no delay from the assignment of the operations order and its conversion to the set of ModSAF unit tasks. The selection of which set of ModSAF tasks sets to represent each operations order phase was arbitrary. Generally, the Mission Planner attempted to mimic the actual company operations by the decomposition of the company phase into a reasonable set of ModSAF tasks.

The ratio of ModSAF tasks to assignable company operations order phases varied from three ModSAF tasks to one phase to two ModSAF tasks to one phase, for an average of 2.5 tasks to one operations order phase. Additionally, mission selection was significantly easier with the company operations order, as the number and types of choices were extremely limited for the human to select.

B. EXAMPLE MISSION ASSIGNMENT

In this example, a tank company is created and assigned a set of company operations order tasks. The operations order is included from the print command as a reference.

1. Unit Creation

The unit must first be created by selecting the unit editor icon from the menu bar on the left of the screen (Figure 16). The unit is placed on the map with the desired formation, orientation, and other data modified as required. The unit must be a ground maneuver company, such as an Abrams (M1) tank company, or a Bradley (M2) infantry company. Company teams consisting of platoons of both tanks and infantry fighting vehicles can also be selected.

2. Unit Selection

Once the company-sized unit is in place on the map, it may be selected using the mouse. The top-level organization icon must be selected to inform ModSAF that a company-level operation is to occur. Once the unit has been selected, a Unit Operations editor will appear, with the unit organization and an execution matrix (Figure 17). An "Operations Order" pushbutton can be found in the lower left corner of the editor. Selecting this will bring up the Operations Order base editor (Figure 18)

3. Operations Order Preparation

The operations order may now be produced. It may be completed in any order; currently the Paragraph One, Two, Four and Five buttons will allow for the input of data, but little is required to convert the Paragraph Three phases to ModSAF tasks. Figure 19 shows the "Paragraph Two: Mission" editor.

The order may be printed at any time to determine its status in preparation. It will print to the window from which ModSAF was first initiated. The format of the operations order closely follows that of an actual five paragraph field order. Figure 20 shows the sample operations order once all information has been completed, while Figure 21 shows the completed Operations Order base editor.

4. Mission Assignment

Assigning the mission will cause a termination of the operations order editor, and a resumption of the unit operations editor. The conversion of the operations order information into a set of ModSAF tasks and enabling tasks can now proceed. The company now has a set of orders, and will execute them according to its current state and the enemy situation (Figure 22).

C. SUMMARY

The mission planner simplified the assignment of company-level tasks. The mission planner can produce a more detailed set of orders in less time using the Operations Order editors than can a skilled ModSAF operator using the Unit Operations editor's execution matrix. On the average, it requires approximately one minute to enter a four-phase company mission using the mission planner; it requires at least twice that time using the Unit Operations execution matrix.

These results, however, belie the true potential of the mission planner's capability to develop realistic and detailed missions for company-level forces. This prototype was simplified by implementing only company-level tasks. These tasks are not very realistic; apparently ModSAF Version 1.3 and later will implement company-level tasks in a more detailed and realistic manner. The fact remains that the platoons do not communicate with each other, and most, if not all, tasks are currently homogeneous in nature. This means that all the platoons are all doing the same task, with little or no coordination and direction at the company level.

This does not mean that the implementation of AAFSMs preclude unit planning and coordination. A parallel work has accomplished this within the ModSAF framework of AAFSM development at the company level [MCAND94]. However, the coordination and communication requirements involved make this approach very cumbersome and prone to the insertion of anomalies.

The lack of real-time coordination among subordinate elements of a company is beyond the scope of this work. However, the use of company-level tasks do little to make the actions of the company more realistic; in fact, just the opposite is true. A viable argument could be made in favor of a different approach at the company level to the use of a mission planner of some form that would select platoon-level tasks and assign them in a coherent and intelligent manner to a company-sized unit to accomplish a particular mission. This would take advantage of the relative strengths of the AAFSM at the platoon and lower levels to efficiently model the individual entities; while the mission planning aspects that begin at company-level can be performed in another manner.

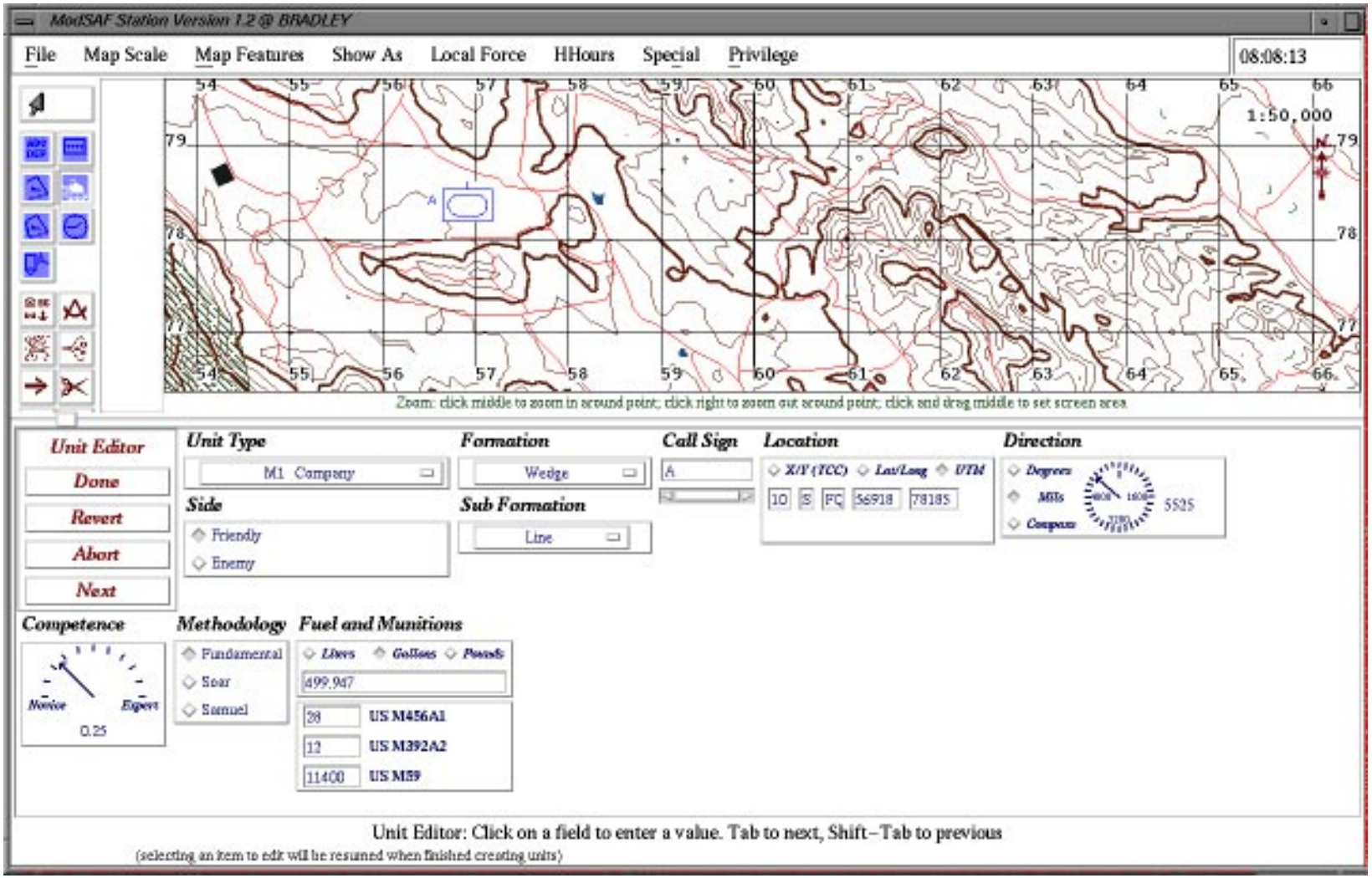


Figure 16: ModSAF Units Creation Editor

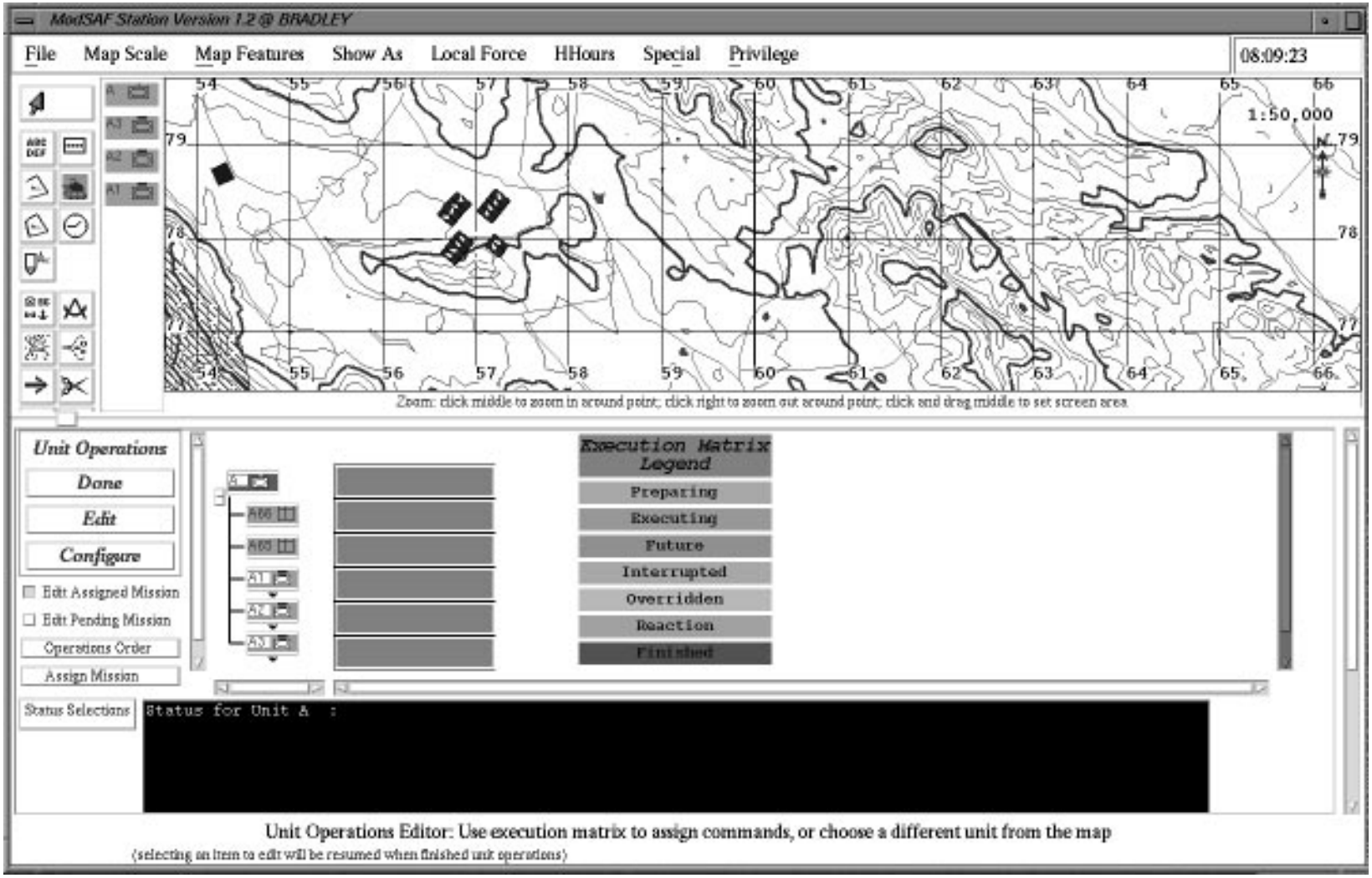


Figure 17: Unit Operations Editor -- Initial View

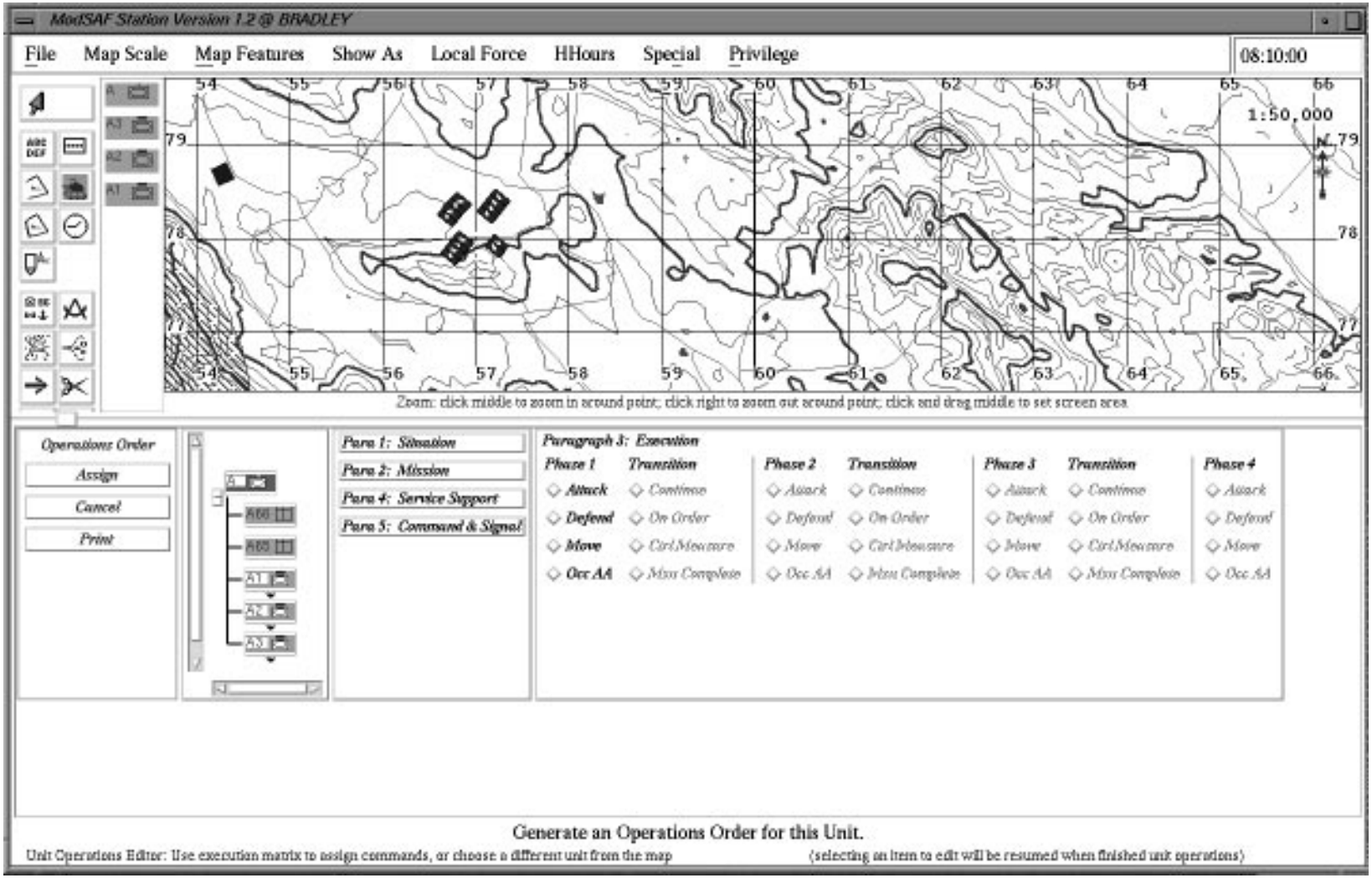


Figure 18: Operations Order Editor -- Initial View

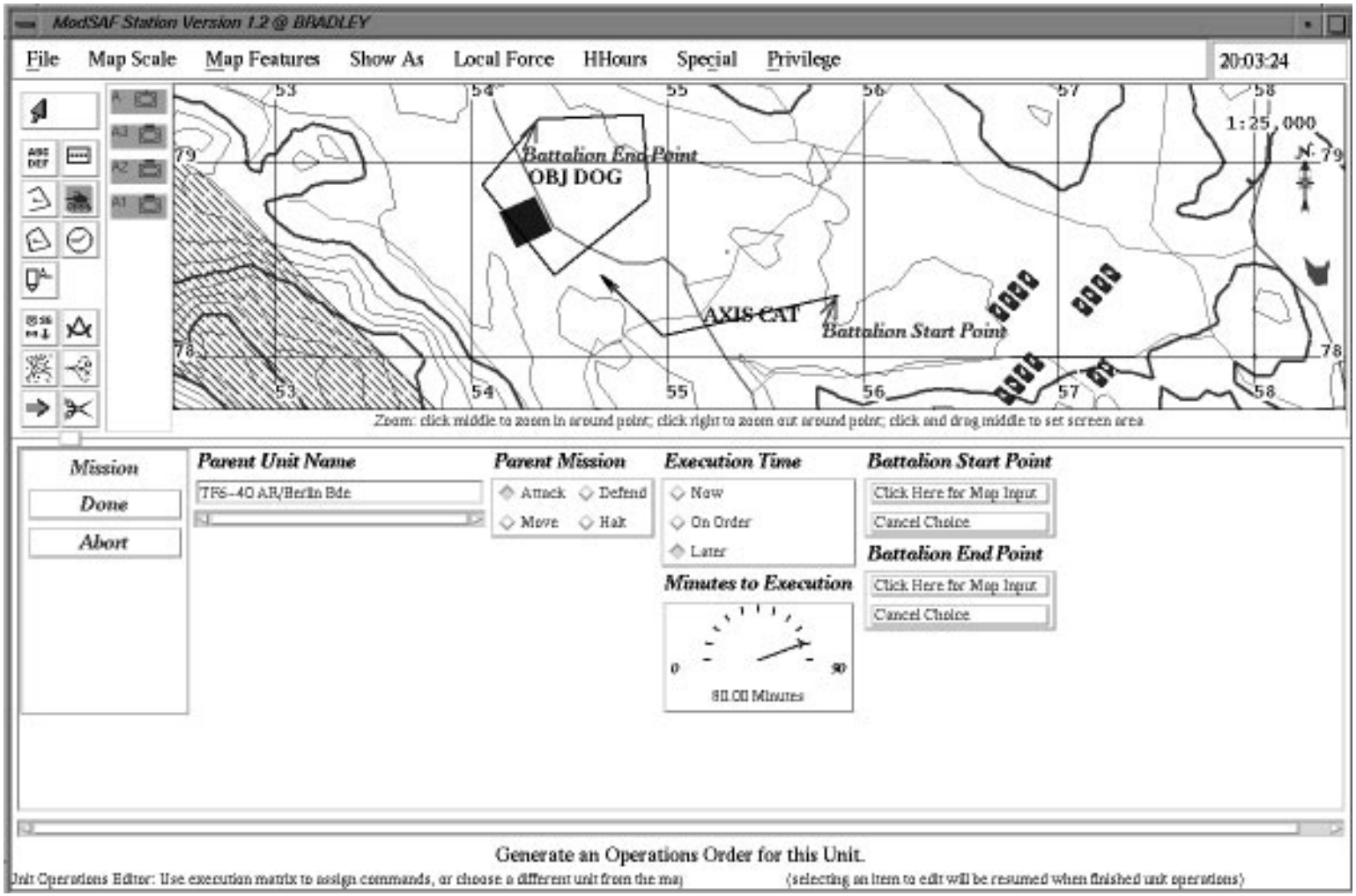


Figure 19: Paragraph Two: Mission Subeditor

Operations Order for A
PO Database Number 1/1/359
Paragraph 1: Situation
a. Enemy forces.
Total Number of Enemy Combat Systems in Area of Interest: 83

b. Friendly Forces.
Total number of Friendly Combat Systems in Area of Interest: 60
Number of Tanks: 14
Number of IFV's: 4
Number of Other Vehicles: 2

c. Area of Interest (Company Level).
Southwest Corner Location: UTM Grid: 10SFQ546773
Southeast Corner Point: UTM Grid: 10SFQ595771
Northwest Corner Point: UTM Grid: 10SFQ542795
Northeast Corner Point: UTM Grid: 10SFQ593795

Paragraph 2: Mission
TF 6-40 AR/Berlin Bde attacks
45 minutes from now, from point UTM Grid: 10SFQ562784 to point UTM Grid: 10SFQ545788

Paragraph 3: Execution
a. Concept of the Operation
This Mission will be executed in 4 phases.
b. Detailed Instructions -- A Company:
Phase 1:
Attack along axis UTM Grid: 10SFQ551782
Assault from attack position, location UTM Grid: 10SFQ547785
to seize objective, UTM Grid: 10SFQ545788
Phase 2:
Transition from phase 1 to phase 2: on order.
Defend battle position UTM Grid: 10SFQ545788
Oriented on the TRP located UTM Grid: 10SFQ538792
Left Limit: UTM Grid: 10SFQ533789
Right Limit: UTM Grid: 10SFQ542795
Phase 3:
Transition from phase 2 to phase 3: on order.
Conduct a road march, Start Point UTM Grid: 10SFQ562784
End at Release Point UTM Grid: 10SFQ573805
Phase 4:
Transition from phase 3 to phase 4: continue.
Occupy Assembly Area at location UTM Grid: 10SFQ584805

Paragraph 4: Service Support
a. Supply
Supplies on hand:
Ammunition basic load: 100.00.
Fuel basic load: 100.00.
Resupply Points:
Ammunition Resupply Point Location: UTM Grid: 10SFQ582780
Fuel Resupply Point Location: UTM Grid: 10SFQ580784
b. Services
Battalion Aid Station Location: UTM Grid: 10SFQ584784
Admin Log Operations Center Location: UTM Grid: 10SFQ580778

Paragraph 5: Command & Signal
a. Signal.
Current CEOI in Effect.
b. Command.
Chain of Command is
Commander
Third Platoon Leader
First Platoon Leader
Second Platoon Leader

Figure 20: Sample Printed Operations Order

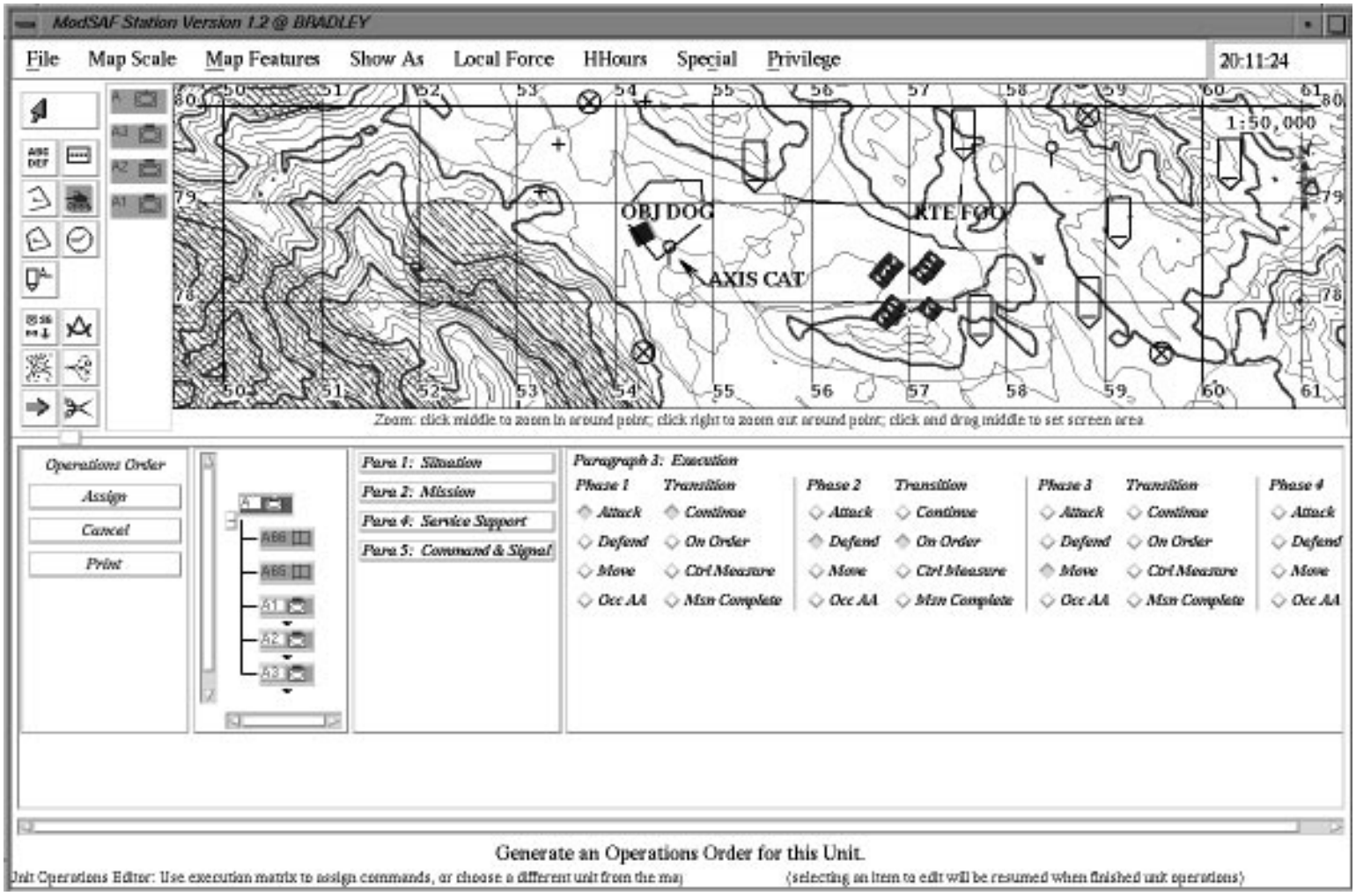


Figure 21: Completed Operations Order Base Editor

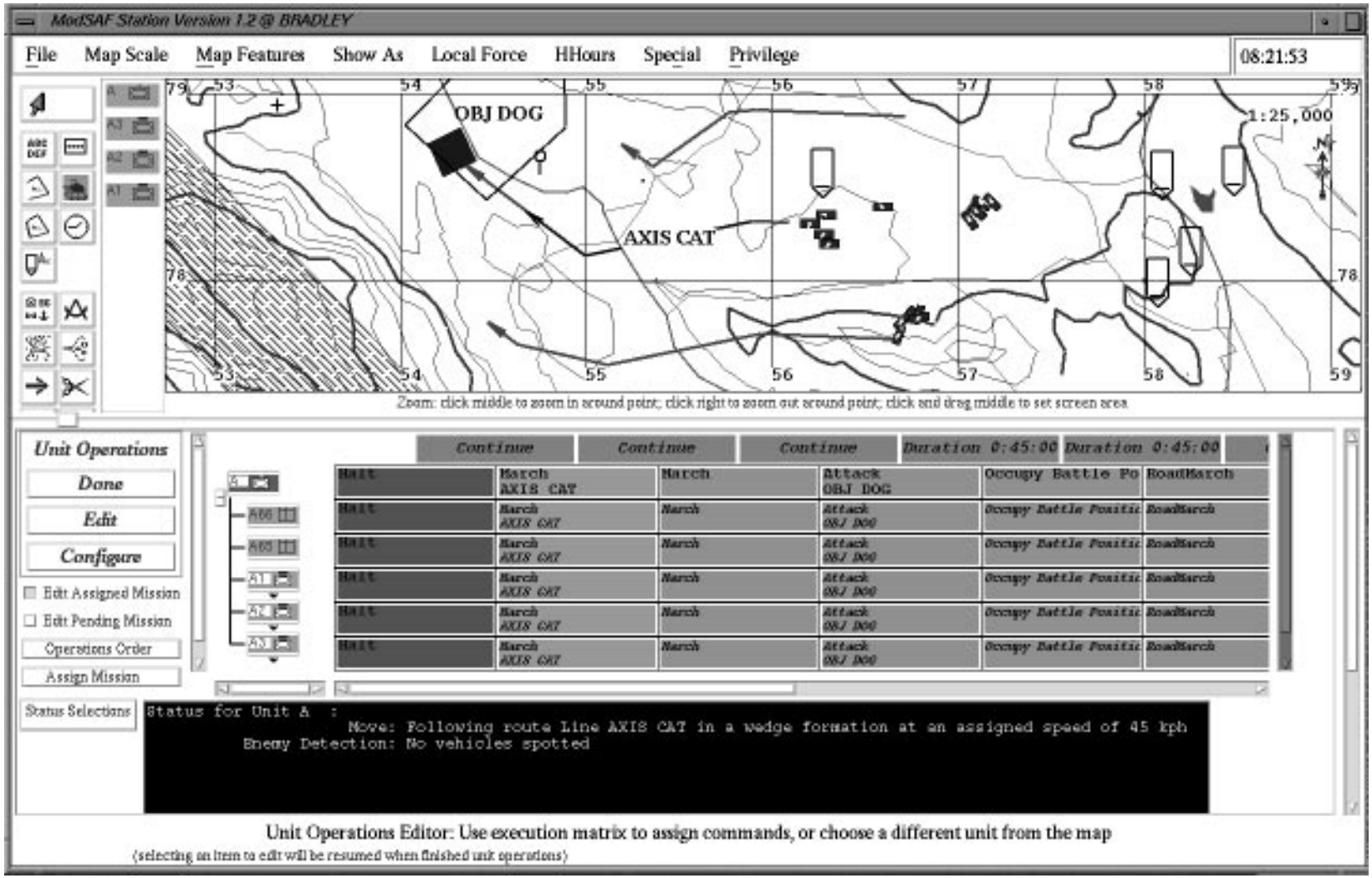


Figure 22: Unit Operations Editor Reflecting Assigned Mission

VIII. SUMMARY AND CONCLUSION

The development of automated planning systems for computer generated forces is still an emerging field. While much work has been done with computer generated forces in designing low-level behaviors and reactions, little has been done to develop and implement a command and control and mission planning architecture; these functions remain the domain of a human expert.

The intent of this mission planner is to abstract the human controller from the tediousness of planning the minutiae of individual vehicles and platoons that comprise a company-level force. The user assumes the role of the Battalion Task Force commander, and the mission planner assumes the role of a company commander. The Task Force commander issues a version of the US Army's five paragraph operations order (OPORD) to a subordinate company, which are represented as ModSAF computer generated forces.

A. MERITS OF THE STRATEGY

While incomplete, the preliminary results of the incorporation of this tool with ModSAF are promising. The average translation of a company operations order phase to a set of ModSAF task frames is 2.5 task frames per operations order phase. Additionally, the mission selections for the company operations order are much simpler, as it is assumed that the artificial intelligence submodules will have the capability to reason about the context of the mission assignment and arrive at similar conclusions. The modularity of the application interface allows for relatively simple insertion and deletion of submodules.

B. SUGGESTED IMPROVEMENTS

The most obvious improvement that could be accomplished is the development of an actual mission planning reasoner. The use of the Rational Behavior Model in computer generated forces command and control has been extensively discussed here, but not fully

implemented. Therefore a full evaluation of this approach cannot be performed until a limited set of the artificial intelligence submodules have been written and integrated with the system. The results of this evaluation may result in the discarding of the integrated development strategy, because the current ModSAF system requires a large amount, too. If the distributed development strategy is re-visited, one should ensure the knowledge in detail of each library in the ModSAF system. Only then will there be a significant advance in the reuse of ModSAF code outside the ModSAF application environment.

C. RECOMMENDATIONS FOR FUTURE WORK

The most pressing need is the development of the Mission Selector/Evaluator module, along with the modification of the current application to support this module. Any artificial intelligence language could be used, however careful consideration must be made of the required data input and output formats. If developed using the integrated development strategy, then performance measures should attempt to determine if there are any measurable performance degradations by ModSAF or by the application. If so, then the distributed strategy needs to be revisited.

The development of a set of heuristics to reduce the search space is needed. This should be developed independently of any language, yet be understandable so that they may be implemented. Strategic-level adversarial planners must use heuristics because the complexity levels without them approach NP-complete. This mission planner shares those traits.

With the increase in complexity levels and capabilities, a single human computer generated forces operator can no longer control large numbers of high-resolution forces during a major exercise. Instead, the user must allow the computer to do the low-level reasoning, providing guidance and information as required. This mission planner prototype could assist the human by providing a framework about which future artificial intelligence research can be conducted

LIST OF REFERENCES

[ARMY85] US Army Field Manual 101-5-1, Operational Terms and Graphics, Headquarters, Department of the Army, October 1985.

[ARMY88] United States Army, Field Manual 71-1, The Tank and Mechanized Infantry Battalion Task Force, Headquarters, Department of the Army, October 1988.

[BRAU93] Braudaway, W., "A Blackboard Approach to Computer Generated Forces," Proceedings of the Third Conference on Computer Generated Forces and Behavioral Representation, University of Central Florida, Orlando, FL, March 17-19 1993, pp 11-20.

[BYRN93] Byrnes, R. B., Nelson, M. L., Kwak, S.,McGhee, R. B., Healey, A. J. "Rational Behavior Model: An Implemented Tri-Level Multilingual Software Architecture for Control of Autonomous Underwater Vehicles," Proceedings of the 8th International Symposium on Unmanned Untethered Submersible Technology, University of New Hampshire, Durham, NH, September 27-29 1993, pp. 160-178.

[CALD93] Calder, R. B., Smith, J. E., Courtemanche, A. J., Mar, J. M. F., Ceranowicz, A. Z., "ModSAF Behavior Simulation and Control," Proceedings of the Third Conference on Computer Generated Forces and Behavioral Representation, University of Central Florida, Orlando, FL, March 17-19 1993.

[CER92] Ceranowicz, A. Z., "ModSAF Programmer's Guide," Naval Research Laboratory, Contract No. N00014-92-C2150, 1992.

[CERa94] Ceranowicz, A., "Modular Semi-Automated Forces," Modular Semi-Automated Forces: Recent and Historical Publications, Loral ADS Document No. 94007 v. 1.0, 1994, pp 1-9.

[CERb94] Ceranowicz, A., "ModSAF and Command and Control," Modular Semi-Automated Forces: Recent and Historical Publications, Loral ADS Document No. 94007 v. 1.0, 1994, pp 11-27.

[DMSO93] Defense Modeling and Simulation Office, "1993 DMSO Survey of Semi-Automated Forces," DMSO Summary Report, 1993.

[DURK94] Durkin, J., Expert Systems Design and Development, Macmillan, 1994.

[ELSA91] Elsaesser, C., MacMillan, T., "Representation and Algorithms for Multiagent Adversarial Planning," The MITRE Corporation, Document # MTR-91W000207, 1991.

[GIARR93] Giarratano, J. C., CLIPS User's Guide, CLIPS Version 6.0, Lyndon B. Johnson Space Center Information Systems Directorate, Software Technology Branch (NASA), 1993.

[LEE94] Lee, J. J., Fishwick, P. A., "Simulation-Based Planning for Computer Generated Forces," Proceedings of the Fourth Conference on Computer Generated Forces and Behavioral Representation, University of Central Florida, Orlando, FL, May 4-6 1994, pp 451-460.

[LORAa93] Loral ADS, "ModSAF Software Architecture Design and Overview Document," 1993.

[LORAb93] Loral ADS, "A Modular Solution for Semi-Automated Forces -- ModSAF, An Overview," Loral ADS Briefing Slides, 1993.

[LORAL94] Loral ADS, "ModSAF User Manual, Version 1.2," 30 June 1994.

[MCAND94] McAndrews, G., "Autonomous Agent Interactions in a Real-Time Simulation System," Master's Thesis, Naval Postgraduate School, Monterey, CA, September 1994.

[MOHN94] Mohn, H. L., Pratt, D. R., McGhee, R. B., "Meta-Level C2/Mission Planning Tool for ModSAF," Proceedings of the Fourth Conference on Computer Generated Forces and Behavioral Representation, University of Central Florida, Orlando, FL, May 4-6 1994, pp 461-472.

[PARS94] Parsons, J. D., "Using Fuzzy Logic Control Technology to Simulate Human Decision-Making in Warfare Models," Proceedings of the Fourth Conference on Computer Generated Forces and Behavioral Representation, University of Central Florida, Orlando, FL, May 4-6 1994, pp 519-529.

[PETTY94] Petty, M. D., "The Turing Test as an Evaluation Criterion for Computer Generated Forces," Proceedings of the Fourth Conference on Computer Generated Forces and Behavioral Representation, University of Central Florida, Orlando, FL, May 4-6 1994, pp 107-116.

[PRATT94] Pratt, D. R., Bhargava, H. K., Culpepper, M., Locke, J., "Collaborative Autonomous Agents in the NPSNET Virtual World," Proceedings of the Fourth Conference on Computer Generated Forces and Behavioral Representation, University of Central Florida, Orlando, FL, May 4-6 1994, pp 177-186.

[SALI93] Salisbury, M., Tallis, H., "Automated Planning and Replanning for Battlefield Simulation," Proceedings of the Third Conference on Computer Generated Forces and Behavioral Representation, University of Central Florida, Orlando, FL, March 17-19 1993, pp 11-20.

[SILIC92] Silicon Graphics, IRIS Indigo Owner's Guide for the R3000 and R4000 Models, Silicon Graphics, 1992.

[SMITHa93] Smith, J. E., "LibPO -- Persistent Object Library," Programmer's Reference Guide, ModSAF Documentation, 1993.

[SMITHb93] Smith, J. E., "Compact Terrain Database Library User Manual and Report," Programmer's Reference Guide, ModSAF Documentation, 1993.

[SMITHc93] Smith, J. E., "LibEditor," Programmer's Reference Guide, ModSAF Documentation, 1993.

[SMITHd93] Smith, J. E., "LibTask," Programmer's Reference Guide, ModSAF Documentation, 1993.

[SMITHe93] Smith, J. E., "LibUnits," Programmer's Reference Guide, ModSAF Documentation, 1993.

[SMITHf93] Smith, J. E., "LibSAFGUI," Programmer's Reference Guide, ModSAF Documentation, 1993.

[SMITHa94] Smith, J. E., Courtemanche, A. J., Coffin, D. A., "ModSAF 1.2 Release Notes," ModSAF Documentation, Loral ADS, 1994.

[SMITHb94] Smith, J. E., "LibTaskUtil," Programmer's Reference Guide, ModSAF Documentation, 1994.

[STAN93] Stanzione, T., Smith, J. E., Brock, D. L., Mar, J. M. F., Calder, R. B., "Terrain Reasoning in the ODIN Semi-Automated Forces System," Proceedings of the Third Conference on Computer Generated Forces and Behavioral Representation, University of Central Florida, Orlando, FL, March 17-19 1993.

[STEE90] Steele, G. L., Common LISP, 2d Ed., Digital Press, 1990.

[STRO91] Stroustrup, B., The C++ Programming Language, 2d Ed., Addison-Wesley, 1991.

[ZADEH65] Zadeh, L. A., "Fuzzy Sets," Information and Control 8, 1965.

[ZADEH79] Zadeh, L. A., "A Theory of Approximate Reasoning," in Hayes, J. E., Michie, D., and Mikulich, L. I. (Eds.), Machine Intelligence 9, Chichester, England: Ellis Horwood Ltd., 1979.

APPENDIX A. OPERATIONS ORDER DESCRIPTION AND EXAMPLE

The following is a brief description of the composition of the US Army's five paragraph field order. Additionally, a description of maneuver overlays is included, as the two are very closely linked; in most circumstances the maneuver overlay is an integral part of the overall orders package.

A. OPERATIONS ORDER

1. Paragraph One: Situation

This paragraph is further subdivided into two parts, and generally sets the context in which the rest of the order is to be analyzed.

a. Enemy Forces

The enemy situation comprises the first part, and should be the best information available to the unit at the time the operations order is prepared. Naturally, this information can be rather vague or misleading, as it depends upon the ability of the friendly forces to collect, analyze, and disseminate intelligence information in a timely manner. Included in the enemy situation is a listing of the location and disposition of all known enemy units within the area of influence (that area which can influence the outcome of the battle, either by an enemy unit's physical presence, or capability to directly affect the accomplishment of the mission).

b. Friendly Forces

The friendly situation is expressed in terms of a brief mission statement of equal/higher units to the unit's front, rear, left, and right as oriented on the enemy forces. If there is no visible "front," then often the cardinal points of the compass (north, south, east, west) are used.

2. Paragraph Two: Mission

This paragraph contains a general, yet concise, statement of the unit's mission. In these terms, this refers to the mission of the battalion. All mission paragraphs are usually very short -- definitely no longer than four sentences. It will answer "who, what, when, where, and why" in a concise manner.

3. Paragraph Three: Execution

As the "meat" of the order, this paragraph is the most complex.[1] This paragraph is divided into several major subparagraphs.

a. Commander's Intent

The Commander's Intent contains a verbatim copy of the higher commander's intent, and a second paragraph with the Battalion Commander's intent. This is an unformatted, concise statement of the commander's overall intent, similar to the following:

> Speed is of the essence in getting to Objective Red. I want to move quickly, bypassing but reporting pockets of enemy resistance in order to inflict maximum damage and surprise on the enemy. I want the scouts to focus on possible enemy counterattack avenues, and report to me immediately if they see large enemy armored formations. When we get to Objective Red, I want the companies to destroy anything and anyone who resists. All command and control centers will be destroyed, and logistic dumps will be secured if possible; destroyed if we cannot safely hold them. Expect a strong enemy counterattack to our bold move. If the deep attack mission by TAC Air is successful, then we will be ordered to exploit that success by mopping up any organized resistance to the north of Objective Red. Here, we need to move deliberately and carefully, lest we fall into an anti-armor ambush. Keep all our supplies tucked in close to our formation; we cannot afford to be separated from our resupply of ammo and fuel.

b. Concept of the Operation

This subparagraph contains a general description of how the battalion will set about accomplishing its mission. Its intent is to provide the subordinate commanders

---

1. In longer orders, the subparagraphs will refer the reader to a set of annexes and appendices. Thus, while the order itself may be short, the annexes can be many pages in length. Of course, such a detailed order is frowned upon for use at relatively low levels of the command hierarchy.

with a framework to understand how their actions will interact and support the battalion's mission.

c. Instructions to Subordinate Units

The next set of subparagraphs are detailed instructions to each subordinate unit by name. In a battalion operations order, this would include each company, separate platoon, and major staff section that comprise the battalion/task force. Each of these paragraphs will contain specific information that usually pertains to just that company/section; the reader will also need to refer to the operations overlay for additional information (such as locations of objectives, phase lines, boundaries, etc.).

d. Coordinating Instructions

The last subparagraph of the Execution paragraph, this subparagraph contains a listing of general tasks that are applicable to *two or more elements of the task force* [ARMY88]. This does not include the command and signal items, which are enumerated in Paragraph Five. The items that are listed in this subparagraph generally relate to the coordination required between the subordinate units and the task force, and other details that differ from the Standard Operating Procedures (SOP).

e. Execution Paragraph Summary

A subordinate commander need only look in two places for a listing of tasks that pertain to him -- the specific unit instructions, and the Coordinating Instructions. The Mission, Commander's Intent, and Concept of the Operation subparagraphs help him understand how his mission fits within the battalion's mission. Since this format is followed throughout the Army, less time is required to quickly determine the specified and implied tasks for a particular unit.

4. Paragraph Four: Service Support

This paragraph is further subdivided into personnel support and logistics support required to accomplish the mission. Typically, these paragraphs contain information con-

cerning personnel replacements, casualty reporting, logistics resupply, and sustainment operations.

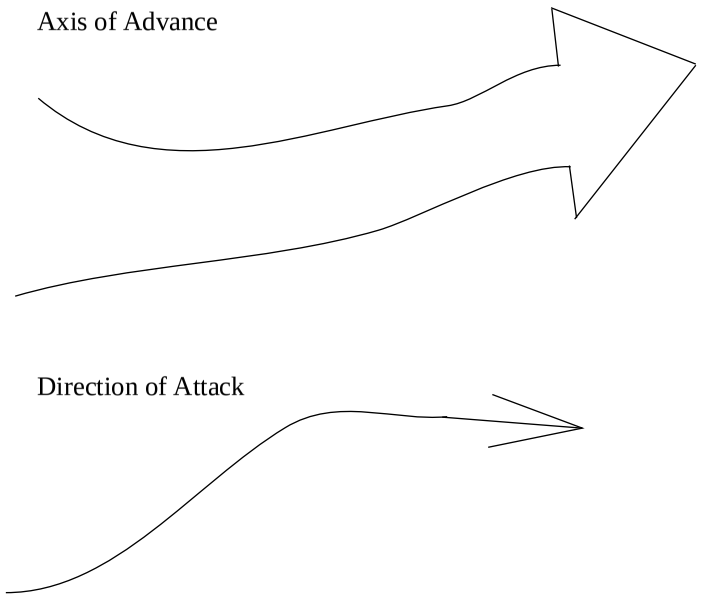
5. Paragraph Five: Command and Signal

This paragraph is further subdivided into two subparagraphs, as denoted by its name. The Command subparagraph details the chain of command so that it is clear who assumes command should the leader be rendered ineffective.The Signal subparagraph details the communications arrangements -- call signs, frequencies, and special event signals such as the use of colored smoke or pyrotechnics.

B. THE OPERATIONS/INTELLIGENCE OVERLAY

The overlays are an essential part of an operations order, in that it is significantly easier to display certain information such as unit locations, control parameters, and operational details in a graphical manner as it relates to a particular point in the world. As a result, the operations and intelligence overlays serve to effectively portray information in a timely manner, and the text portion of the order serves to lend detail and meaning to the graphics.

The military symbols and graphic control measures have specific meaning in their own right. Each graphic control measure has a name, and a definition associated with that name (Figure A-1) [ARMY85].



Axis of Advance -- A general route of advance, assigned for purposes of control, which extends toward the enemy. An axis of advance symbol graphically portrays a commander's intention, such as avoidance of built-up areas or envelopment of an enemy force. It follows terrain suitable for the size of the force assigned the axis and is often a road, a group of roads, or a designated series of locations. A commander may maneuver his forces and supporting fires to either side of an axis of advance provided the unit remains oriented on the axis and the objective. Deviations from an assigned axis of advance must not interfere with the maneuver of adjacent units without prior approval of the higher commander. Enemy forces that do not threaten security or jeopardize mission accomplishment may be bypassed. An axis of advance is not used to direct the control of terrain or the clearance of enemy forces from specific locations. Intermediate objectives normally are assigned for these purposes.

Direction of Attack -- A specific direction or route that the main attack or the main body of the force will follow. If used, it is normally at battalion and lower levels. Direction of attack is a more restrictive control measure than the axis of advance, and units are not free to maneuver off the assigned route. It usually is associated with infantry units conducting night attacks or units involved in limited visibility operations and in counterattacks.

Figure A-1: Example Comparison and Definition of Two Graphic Control Symbols, after [ARMY85]

APPENDIX B. FUZZY SET MEMBERSHIP

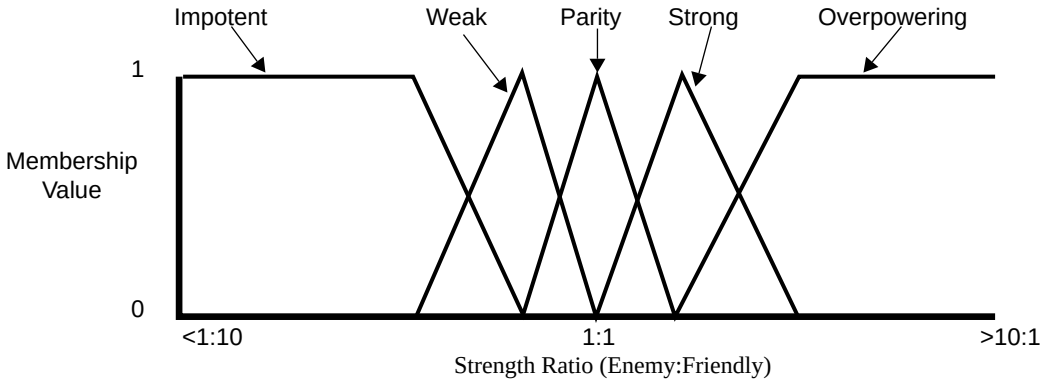
The following figures are membership graphs for the fuzzy sets defined in Chapter V. These graphs contain fit vectors that help ensure that the entire range of values of the fuzzy set is defined by a modifying adjective. This ensures that a fuzzy value will result from any particular "crisp" value, as long as it falls within the range of values. For convenience, Table 7 is repeated below as Table B1.

Enemy Forces	Intelligence Accuracy	Troops Available	Time Available	Terrain Slope
Impotent	Erroneous	Ineffective	None	Level
Weak	Inaccurate	Weak	Short	Gentle
Parity	Questionable	Company Minus	Moderate	Moderate
Strong	Accurate	Normal	Long	Steep
Overpowering	Reliable	Reinforced	Extended	Precipitous

Terrain Soil Type	Terrain Vegetation	Terrain Trafficability	Terrain Obstacles	Success Prediction
Improved	Open	Impassable	Zero	Zero
Normal	Thin	Difficult	Light	Problems
Difficult	Moderate	Moderate	Moderate	Maybe
Impassable	Thick	Easy	Dense	Good
	Dense	Smooth	Impassable	Outstanding

Table B-1: Linguistic Variables and their Modifying Adjectives

Each figure contains a small table that numerically depicts the starting range and ending range of each modifying adjective. The potential y-values are 0 or 1; the x-values denote the ranges.



Impotent

X	Y
1:10	1
1:5	1
1:3	0

Weak

X	Y
1:5	0
1:3	1
1:1	0

Parity

X	Y
1:3	0
1:1	1
3:1	0

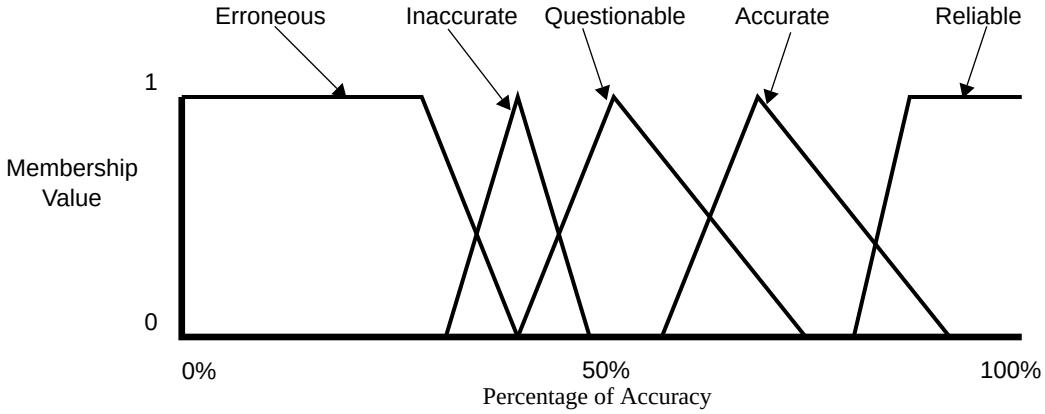
Strong

X	Y
1:1	0
3:1	1
5:1	0

Overpowering

X	Y
3:1	0
5:1	1
10:1	1

Figure B-1: Fuzzy Sets on Enemy Forces



Erroneous

X	Y
0	1
30	1
40	0

Inaccurate

X	Y
30	0
40	1
50	0

Questionable

X	Y
40	0
50	1
70	0

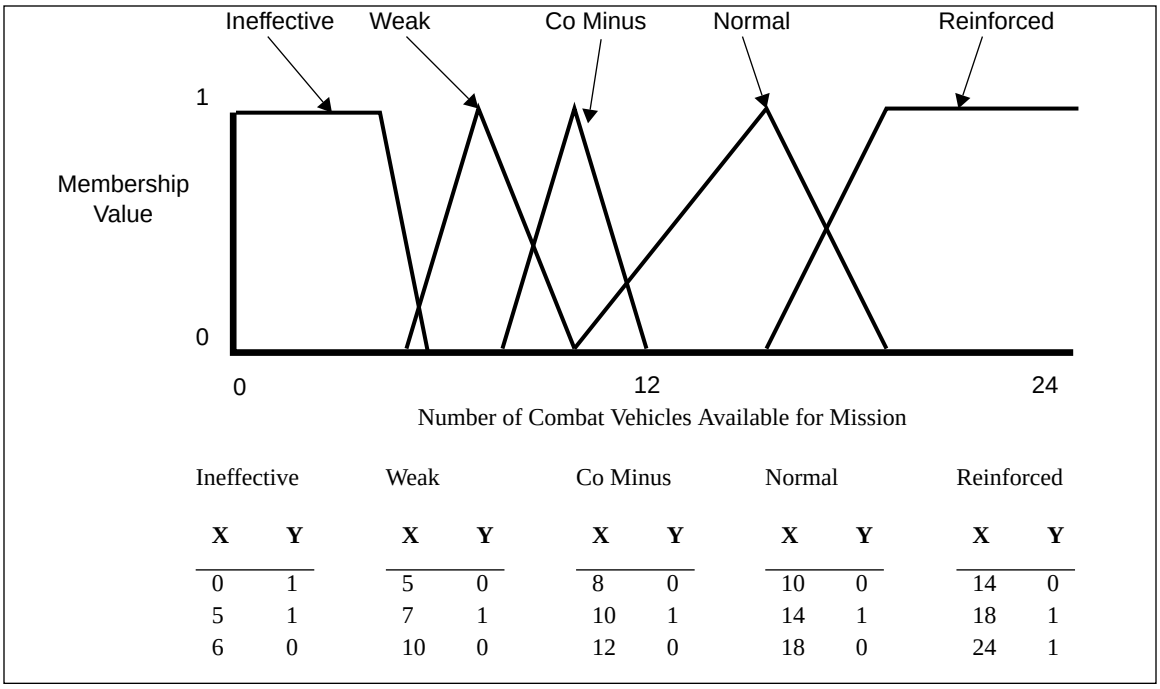
Accurate

X	Y
60	0
70	1
90	0

Reliable

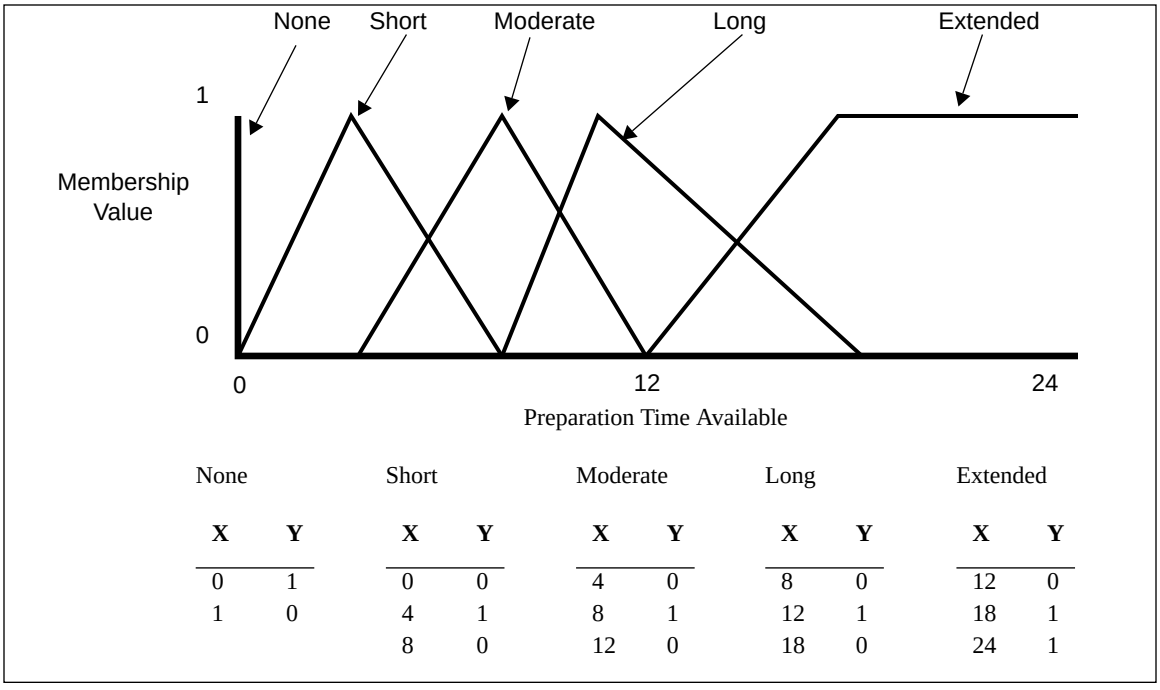
X	Y
80	0
90	1
100	1

Figure B-2: Fuzzy Sets on Intelligence Accuracy



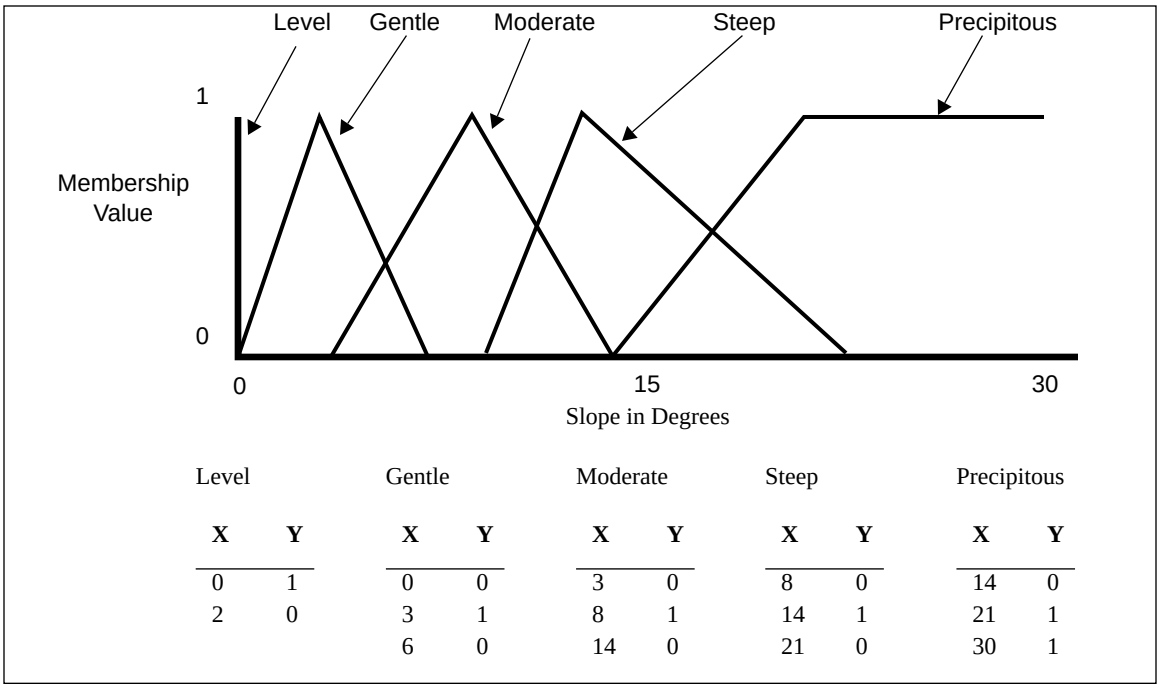
Ineffective		Weak		Co Minus		Normal		Reinforced	
X	**Y**	**X**	**Y**	**X**	**Y**	**X**	**Y**	**X**	**Y**
0	1	5	0	8	0	10	0	14	0
5	1	7	1	10	1	14	1	18	1
6	0	10	0	12	0	18	0	24	1

Figure B-3: Fuzzy Sets on Troops Available



None		Short		Moderate		Long		Extended	
X	**Y**	**X**	**Y**	**X**	**Y**	**X**	**Y**	**X**	**Y**
0	1	0	0	4	0	8	0	12	0
1	0	4	1	8	1	12	1	18	1
		8	0	12	0	18	0	24	1

Figure B-4: Fuzzy Sets on Time Available



Level

X	Y
0	1
2	0

Gentle

X	Y
0	0
3	1
6	0

Moderate

X	Y
3	0
8	1
14	0

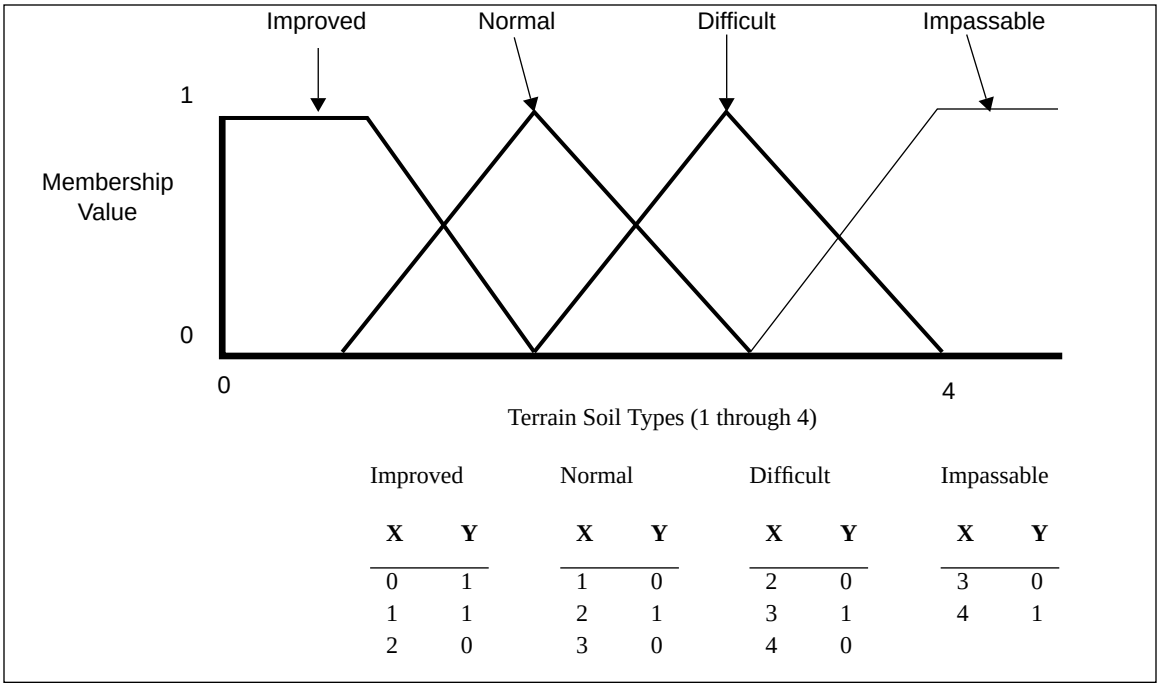
Steep

X	Y
8	0
14	1
21	0

Precipitous

X	Y
14	0
21	1
30	1

Figure B-5: Fuzzy Sets on Terrain Slope



Improved

X	Y
0	1
1	1
2	0

Normal

X	Y
1	0
2	1
3	0

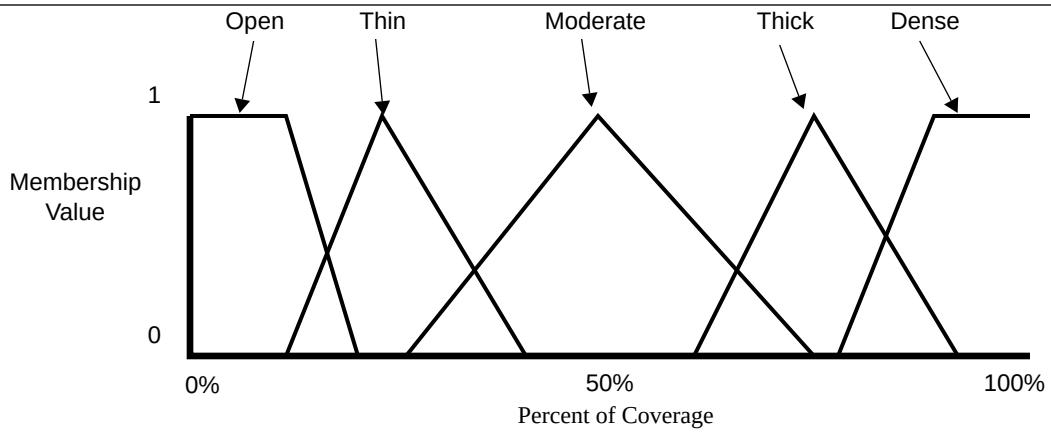
Difficult

X	Y
2	0
3	1
4	0

Impassable

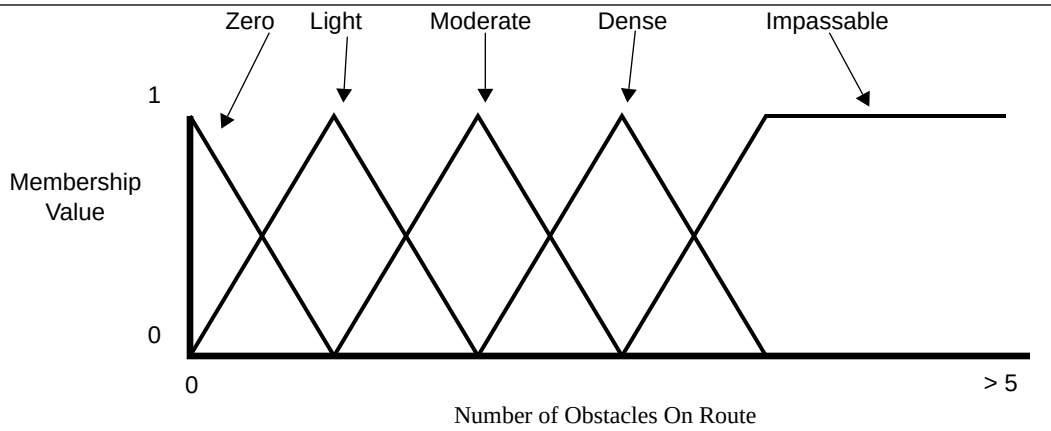
X	Y
3	0
4	1

Figure B-6: Fuzzy Sets on Terrain Soil Type



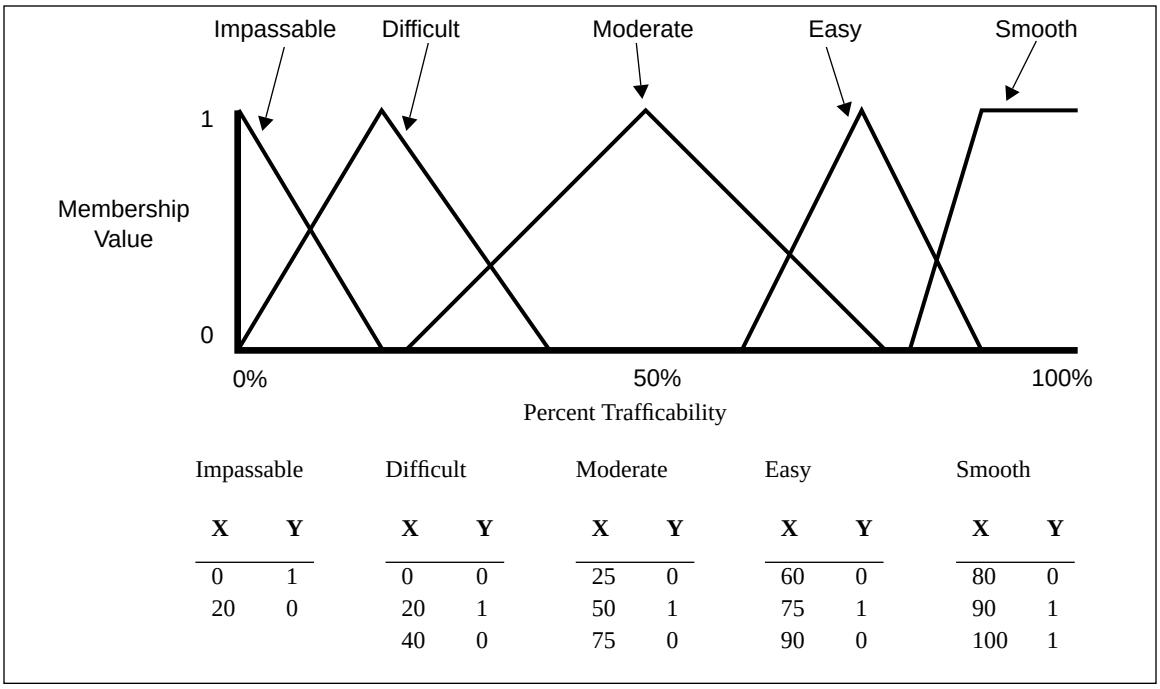
Open		Thin		Moderate		Thick		Dense	
X	**Y**	**X**	**Y**	**X**	**Y**	**X**	**Y**	**X**	**Y**
0	1	10	0	25	0	60	0	80	0
10	1	25	1	50	1	75	1	90	1
20	0	40	0	75	0	90	0	100	1

Figure B-7: Fuzzy Sets on Terrain Vegetation



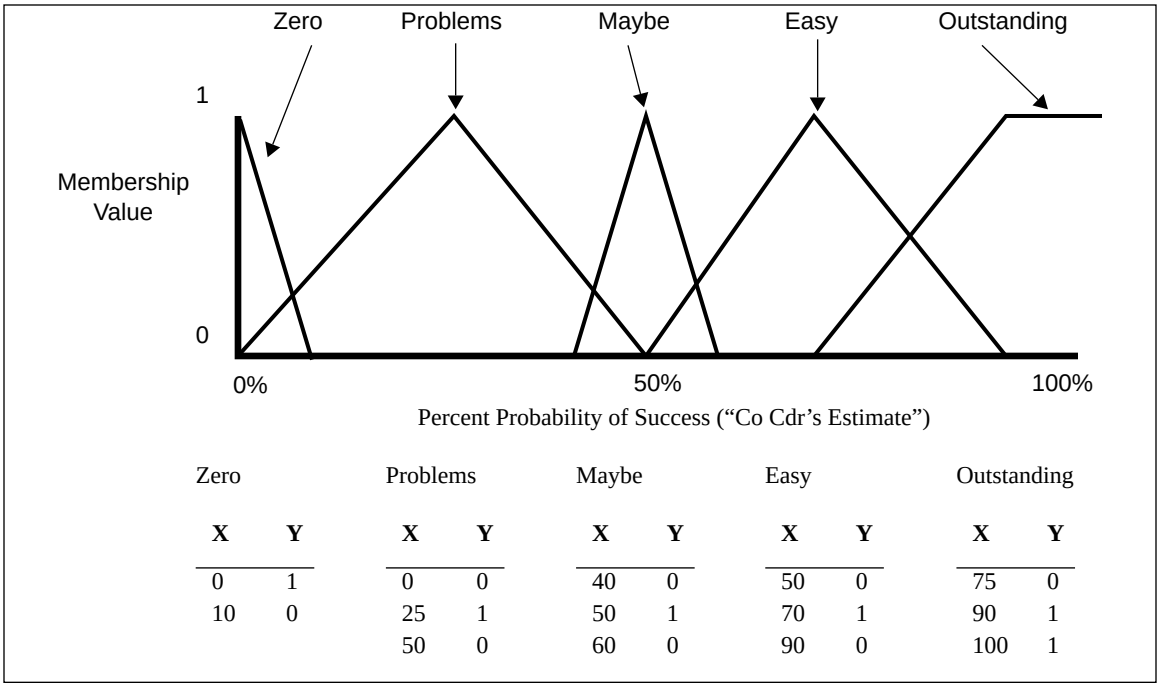
Zero		Light		Moderate		Dense		Impassable	
X	**Y**	**X**	**Y**	**X**	**Y**	**X**	**Y**	**X**	**Y**
0	1	0	0	1	0	2	0	3	0
1	0	1	1	2	1	3	1	4	1
		2	0	3	0	4	0	5	1

Figure B-8: Fuzzy Sets on Terrain Obstacles



Impassable		Difficult		Moderate		Easy		Smooth	
X	**Y**	**X**	**Y**	**X**	**Y**	**X**	**Y**	**X**	**Y**
0	1	0	0	25	0	60	0	80	0
20	0	20	1	50	1	75	1	90	1
		40	0	75	0	90	0	100	1

Figure B-9: Fuzzy Sets on Terrain Trafficability



Zero		Problems		Maybe		Easy		Outstanding	
X	**Y**	**X**	**Y**	**X**	**Y**	**X**	**Y**	**X**	**Y**
0	1	0	0	40	0	50	0	75	0
10	0	25	1	50	1	70	1	90	1
		50	0	60	0	90	0	100	1

Figure B-10: Fuzzy Sets on Success Prediction

APPENDIX C. MISSION PLANNER USER'S GUIDE

The following instructions assume that the Mission Planner is integrated with ModSAF, Version 1.2. The instructions are intended to step the user through the mission planner interactive process, from selecting the company through mission assignment.

A. UNIT SELECTION/OPORD SELECTION

Select a company-sized ground unit from the Plan View Display. If the company has not been created, then use the Unit Creation editor first, and create the company. The Unit Operations editor will appear. It contains information on the status of the company, along with its organization chart, and an execution matrix that shows the status of tasks assigned to the company and its subordinate vehicles and platoons (Figure C-1).

Ensure that the company-level unit symbol at the top of the Unit Organization chart is highlighted. If another vehicle or platoon within the company is highlighted, select the company unit symbol at the top of the Unit Organization chart. If this is not done, the Operations Order editor will not appear, and an error message will appear instead.

Select the "Operations Order" button with the mouse. This will cause the Unit Operations screen to disappear, and the Operations Order base editor to appear in its place.

B. OPERATIONS ORDER BASE EDITOR

The Operations Order base editor consists of four major areas: the editor control buttons, the Unit Organization chart, a set of pushbuttons for Paragraph selection, and Paragraph Three -- Execution (Figure C-2).

1. Editor Control Buttons

These buttons allow the user to assign the operations order to the selected unit, cancel the operations order and return to the Unit Operations editor without action, or to print the operations order to the ModSAF text window.

2. Unit Organization Chart

This currently is for information purposes only; no actions are associated with selecting a subelement of the company. Future versions could include changing the task organization by allowing the insertion and deletion of other subelements in the company.

3. Paragraph Selections

The user can select Paragraphs One, Two, Four, and Five in this section. Each of these pushbuttons opens an editor specific to that paragraph.

a. Paragraph One -- Situation

This editor allows input of data regarding the enemy and friendly situation. This is a simplified model that asks for numbers of friendly and enemy combat systems, and allows for the definition of the Battalion Area of Interest (Figure C-3). The friendly and enemy combat systems numbers allow for determination of force ratios, while the area of interest will be used to constrain the search space for the terrain reasoner.

b. Paragraph Two -- Mission

This editor allows the user to enter the battalion's name and organization, along with the overall mission (Figure C-4). The user may also define the number of minutes to wait before executing the mission. Finally, the user can enter a battalion-level start point and end point. All of this provides background information to the reasoner to allow it to determine the context in which it selects the tasks for the company to execute.

c. Paragraph Four -- Service Support

Resupply points and initial states of supply are defined here (Figure C-5). The use of the ammunition and fuel status allows the user to enter values in percentage of supplies on hand. Resupply points and Administrative-Logistics Operations Center locations can also be defined here.

d. Paragraph Five -- Command and Signal

The company chain of command is defined here (Figure C-6). This will be used by the Mission Selector/Evaluator to determine subunit assignments -- the highest in the chain of command is assumed to be the most competent, and therefore will be assigned the most difficult missions. The signal subparagraph is simply text input for now.

4. Paragraph Three -- Execution

This paragraph is essentially an execution matrix for one company. Therefore, it can be viewed as the specific unit instruction for that company. It allows the user to enter missions for a maximum of four phases. Four basic missions may be selected: Attack, Defend, Move (Road March), and Occupy Assembly Area. Three transitions are allowed: Continue, On Order, and Control Measure. A fourth transition, Mission Complete, allows the user to arbitrarily end the mission assignment at that task. This deselects all missions after the Mission Complete transition.

Each of the missions and the Control Measure transition have their own editor. This allows the user to enter the required data to execute the mission.

a. Attack

This editor requires the user to enter the objective, and the attack position from which to assault the objective (Figure C-7). The axis of advance selection is optional.

b. Defend

This editor requires the input of the battle position to occupy, along with the left, right, and engagement area target reference points (TRP) (Figure C-8). There are no optional entries.

c. Move

This editor requires the input of the start point (SP) and the release point (RP) (Figure C-9). The route selection is optional.

d. Occupy Assembly Area

This editor has one required input -- the center of mass location for the company's assembly area location (Figure C-10).

e. Control Measure Transition

The Control Measure transition has its own editor, which allows the user to select a control measure (such as a phase line) from the Plan View Display as the transition.

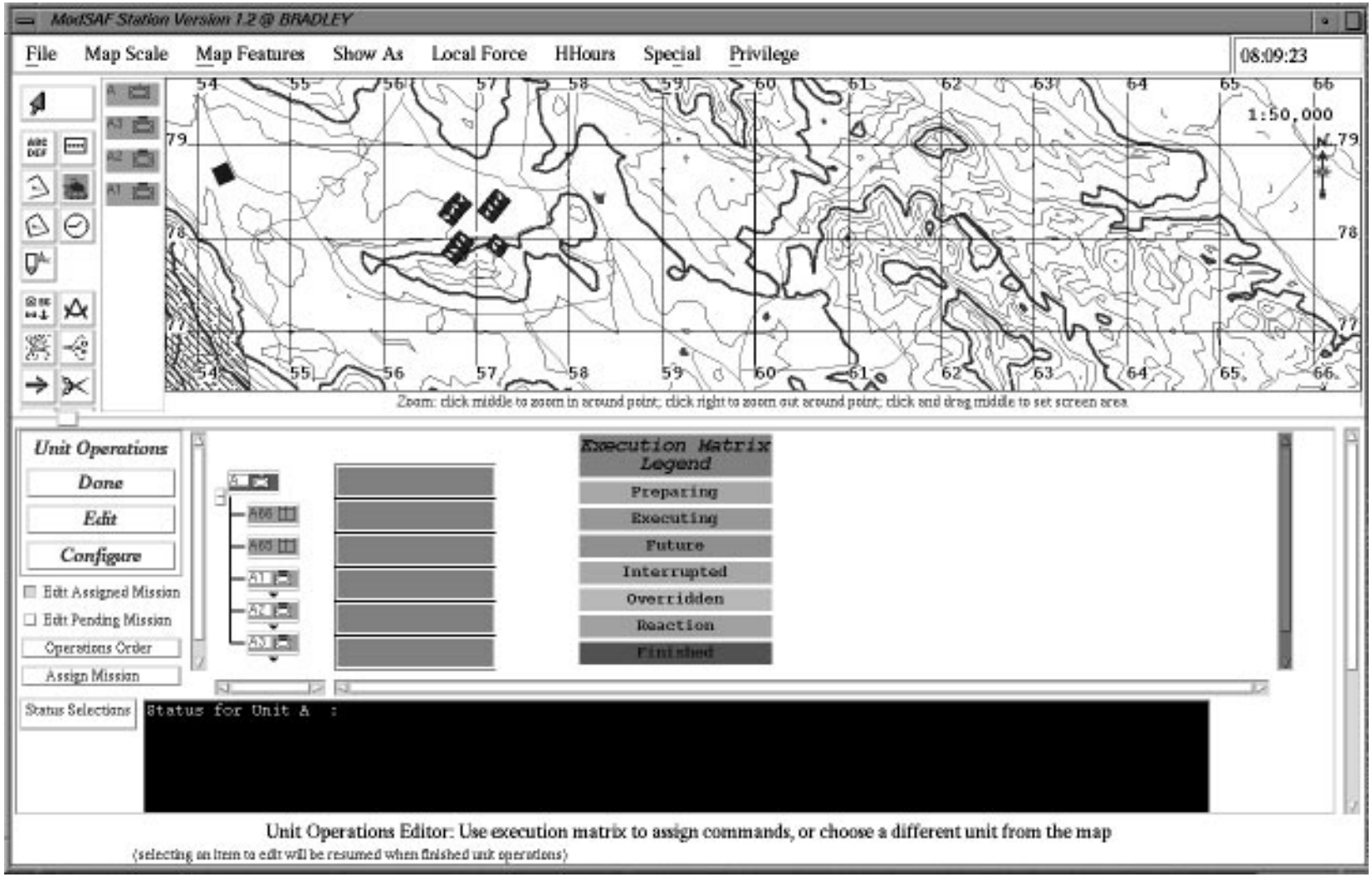


Figure C-1: Unit Operations Editor

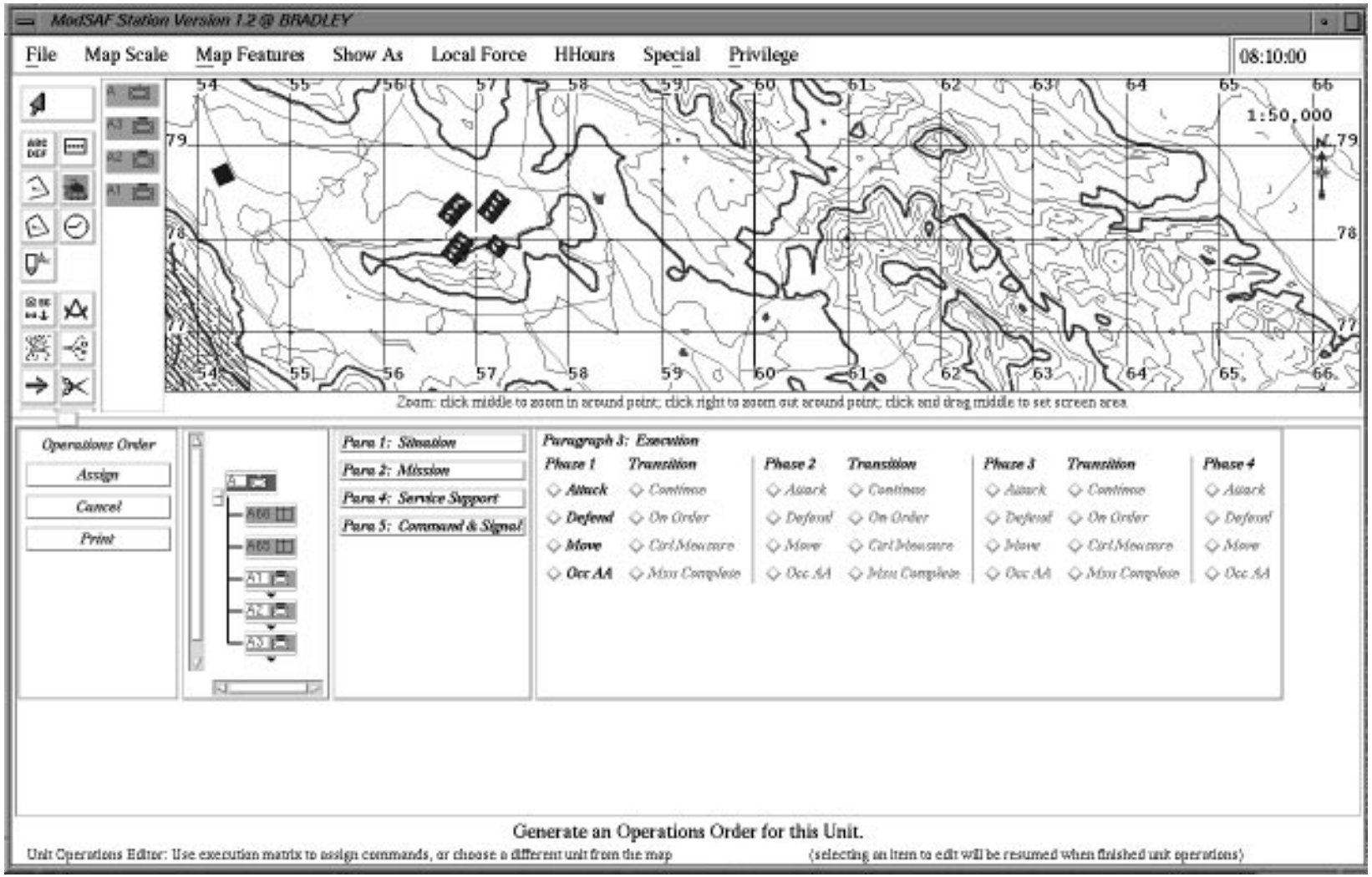


Figure C-2: Operations Order Base Editor

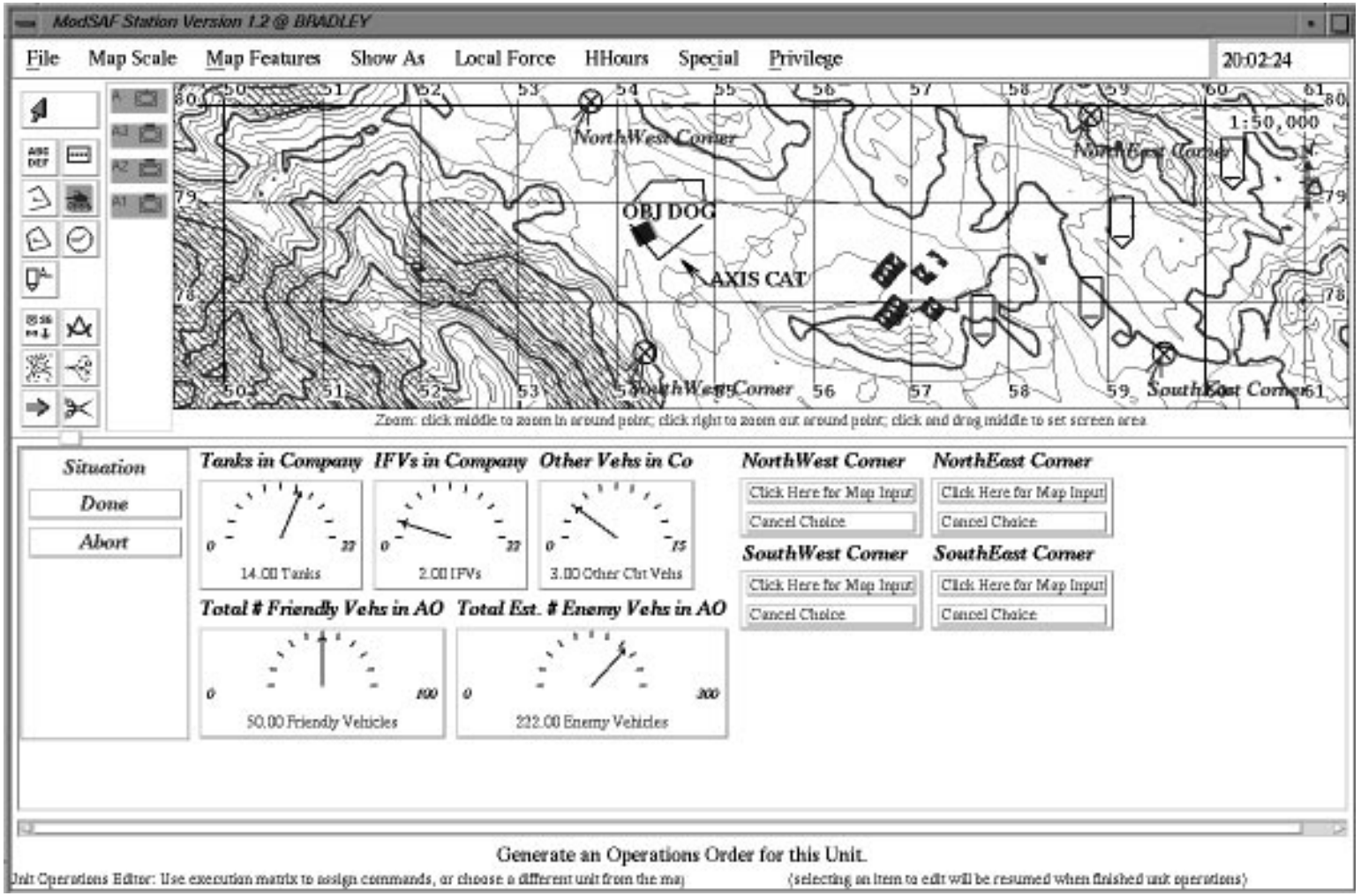


Figure C-3: Paragraph One -- Situation

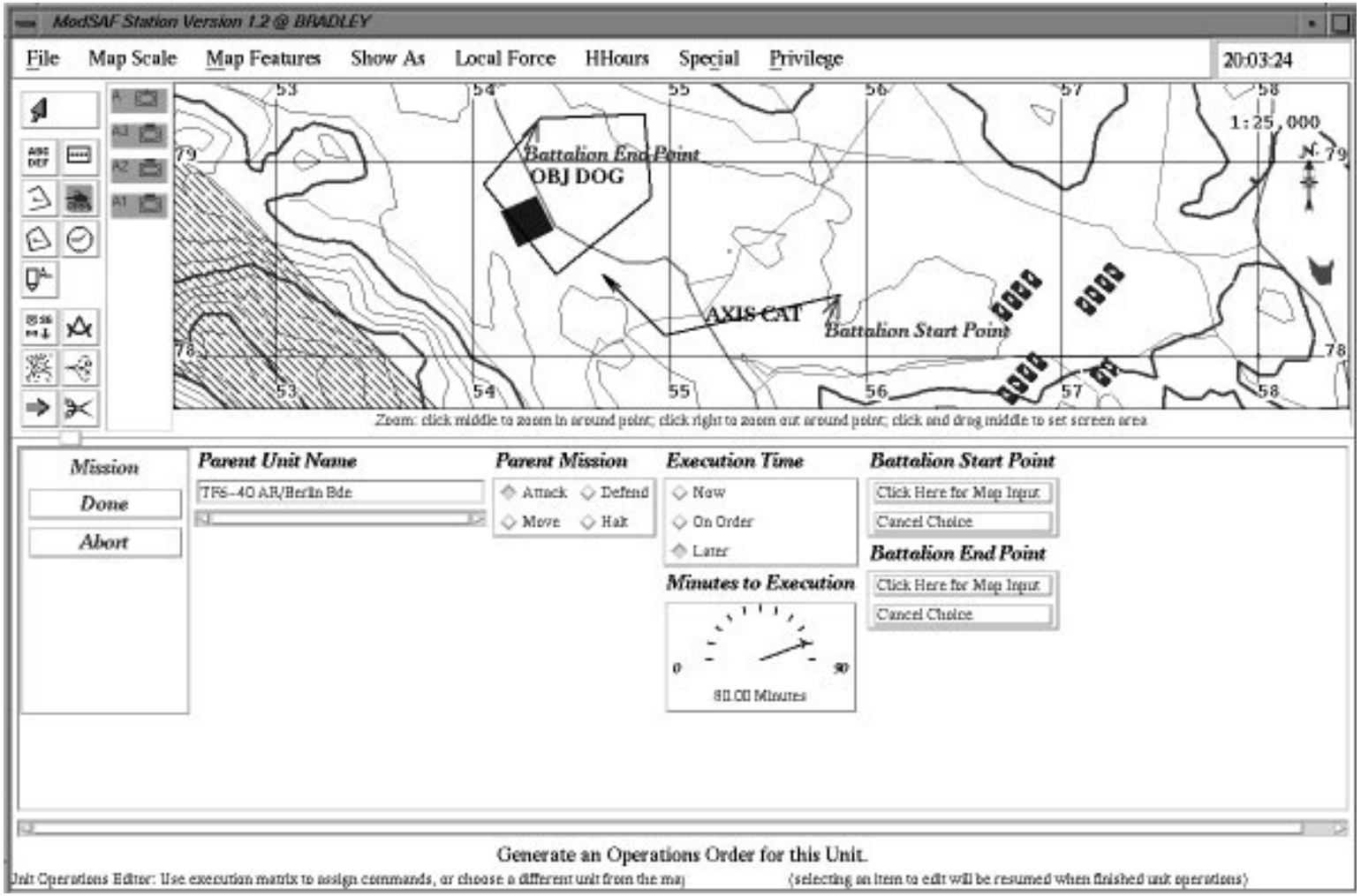


Figure C-4: Paragraph Two -- Mission

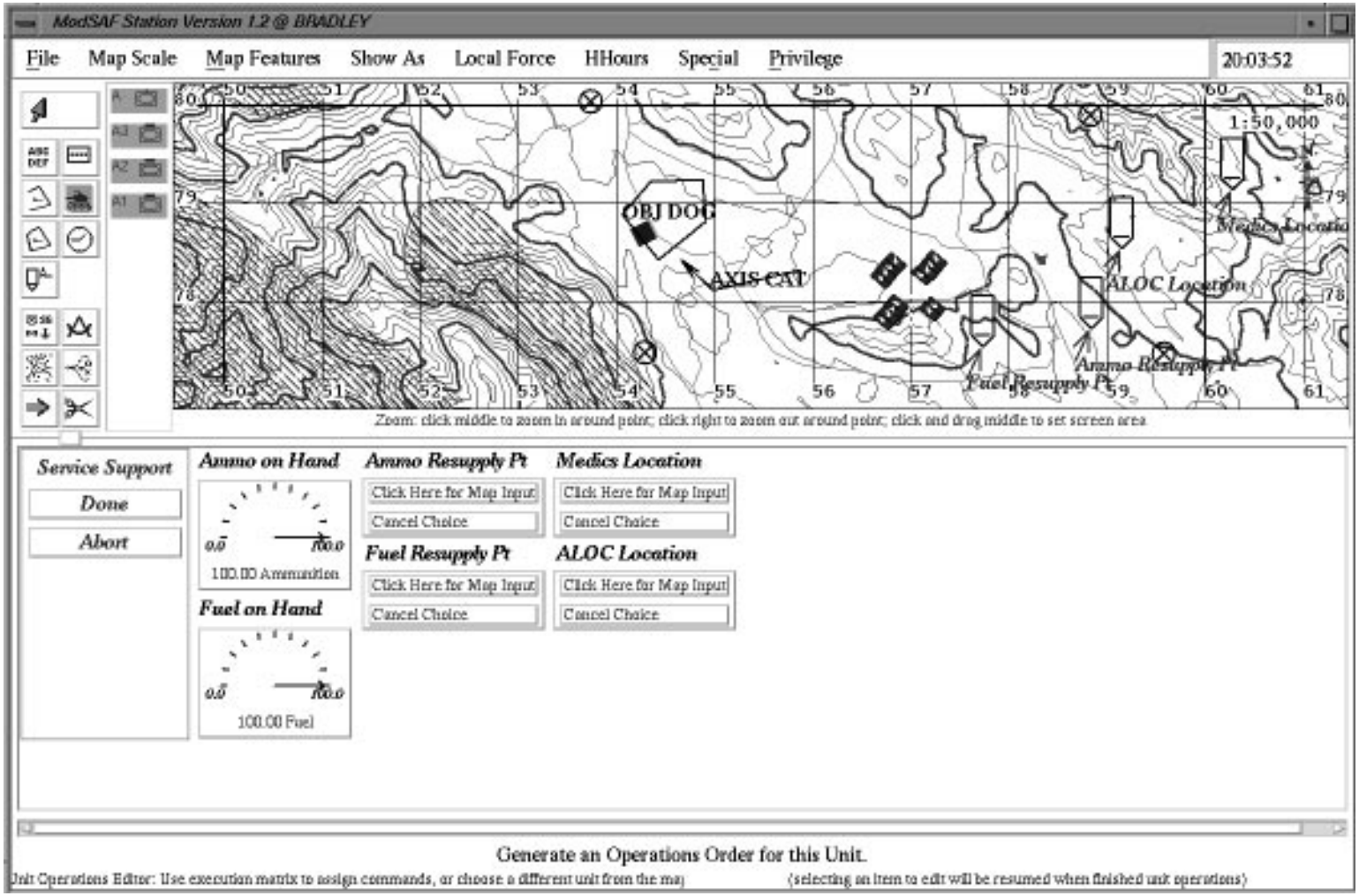


Figure C-5: Paragraph Four -- Service Support

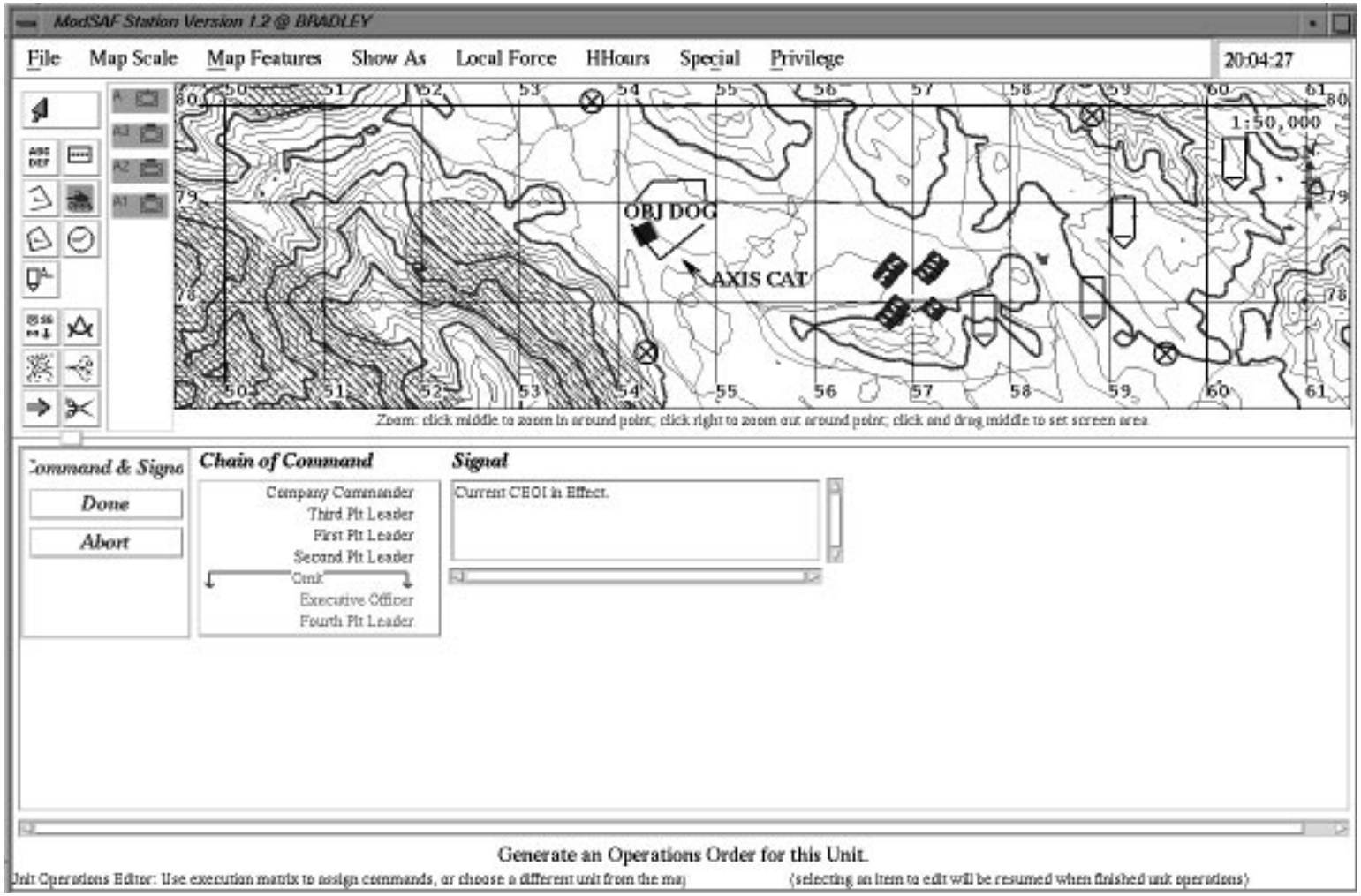


Figure C-6: Paragraph Five -- Command and Signal

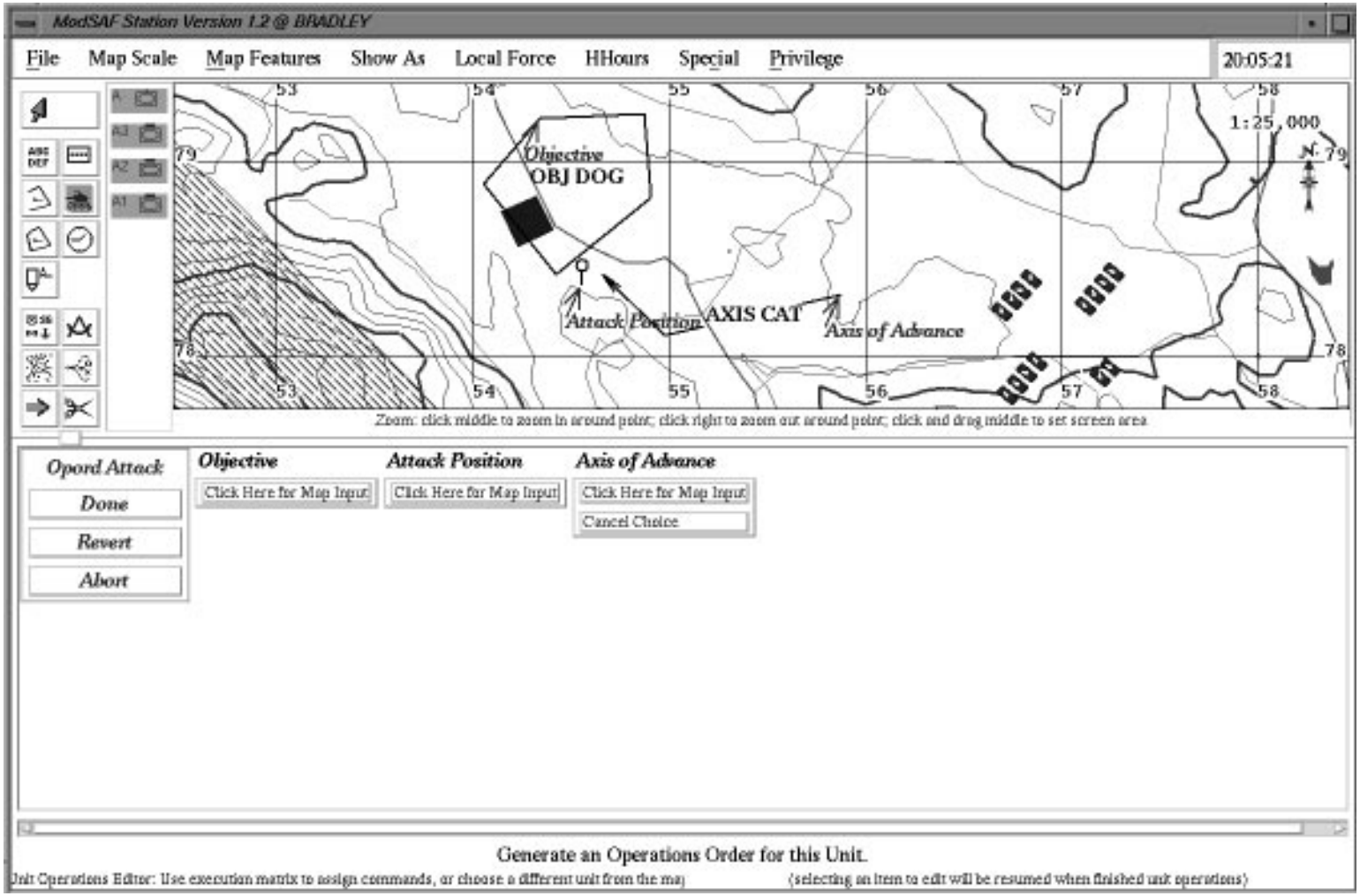


Figure C-7: Attack Mission Editor

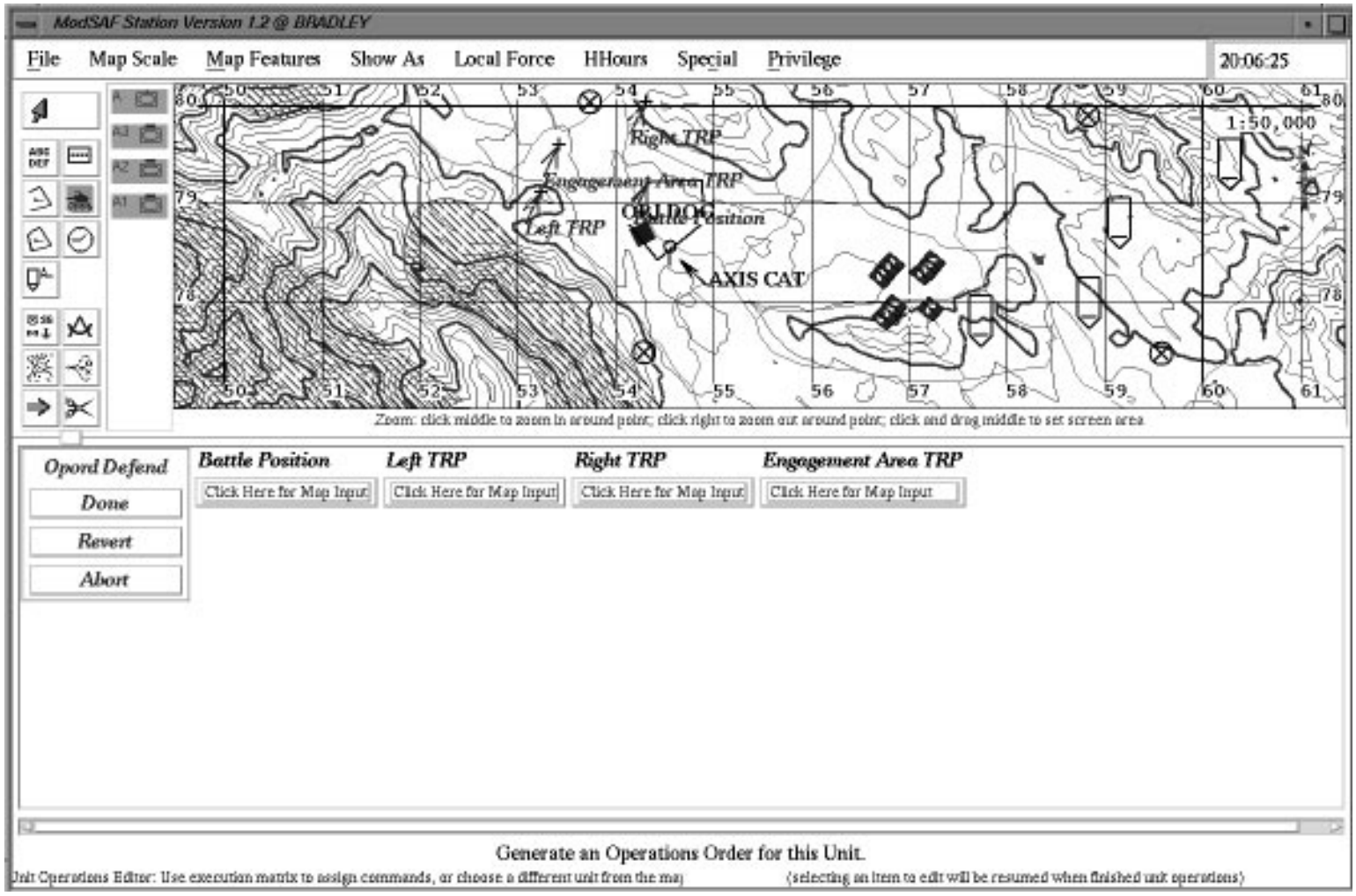


Figure C-8: Defend Mission Editor

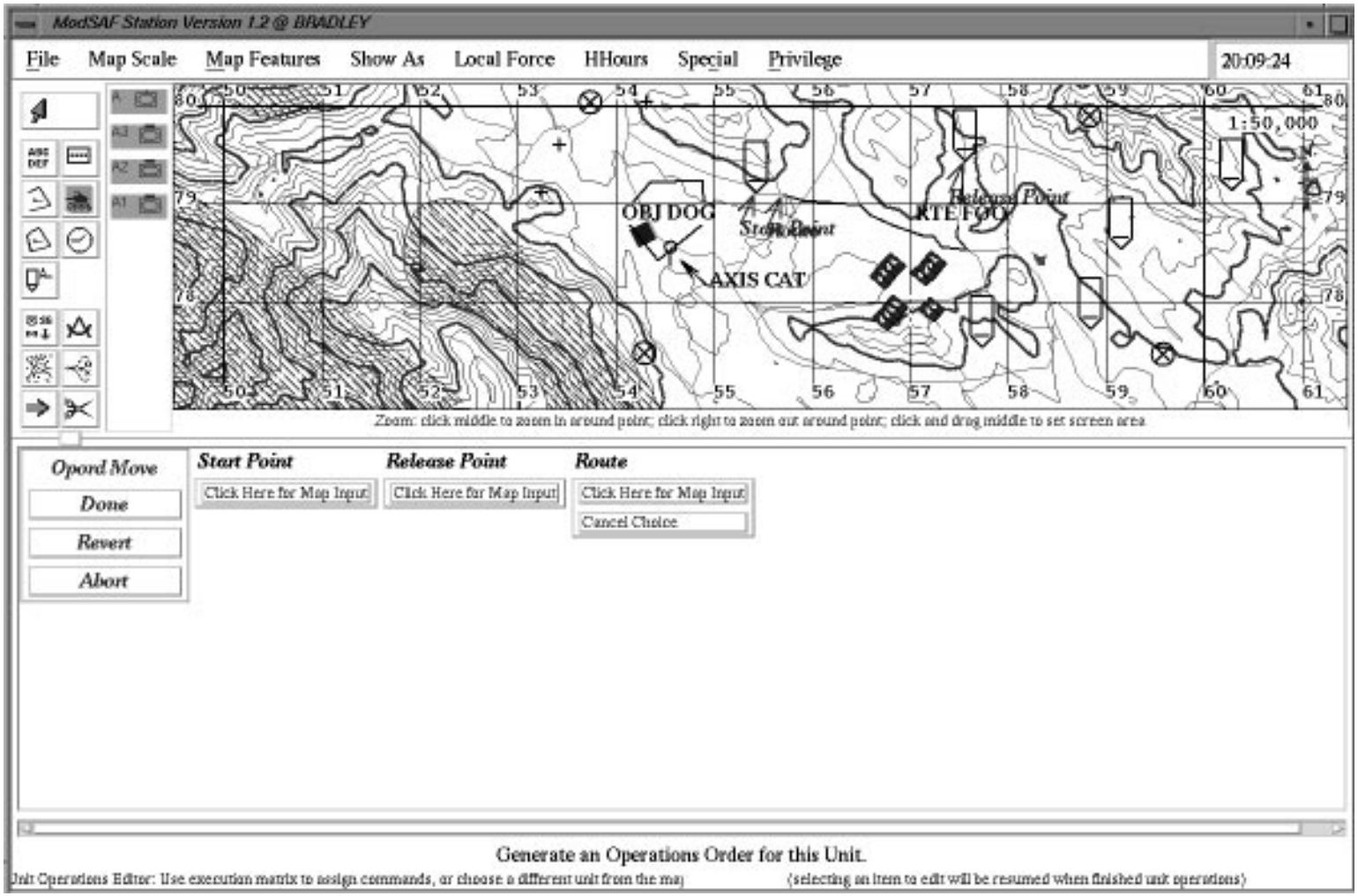


Figure C-9: Move (Road March) Mission Editor

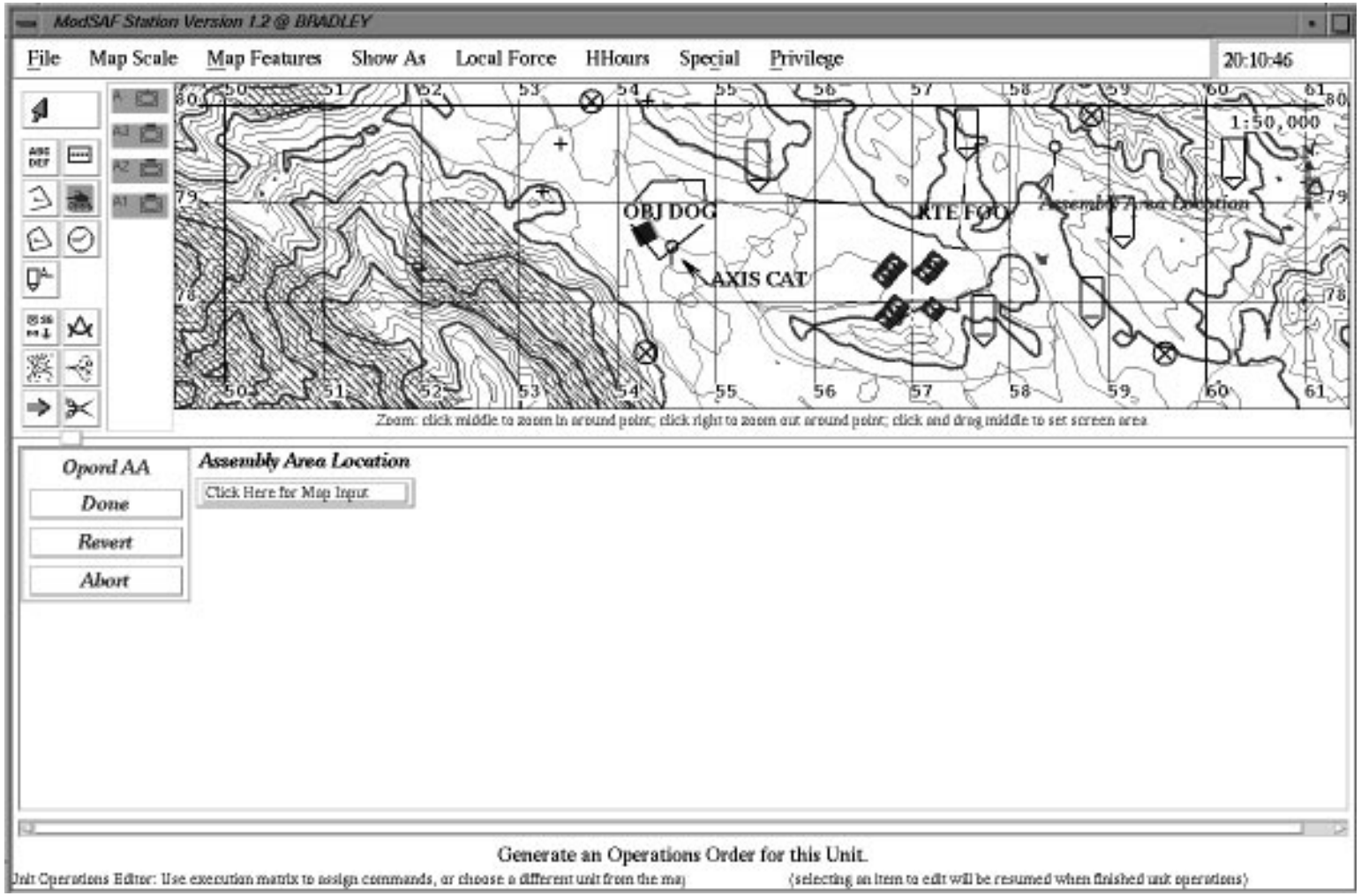


Figure C-10: Assembly Area Mission Editor

APPENDIX D. MISSION PLANNER PROGRAMMER'S GUIDE

This appendix provides information on using the functions associated with the mission planner. The library name in ModSAF is "LibOpord," and all future references will use this name. The structure of this appendix mirrors that of the references for the other ModSAF libraries, to ensure compatibility with ModSAF programming in general.

A. OVERVIEW

LibOpord provides a graphical user interface to a mission planner for company-level ground units. It also provides the framework about which artificial intelligence modules for mission planning may be inserted to provide a means of non-real time planning capabilities at the company level.

LibOpord is closely linked to one library in ModSAF -- LibUnits. The initialization routines and callback functions are all contained within this library. LibOpord is called from the LibUnits Unit Operations editor. Upon completion of its tasks, it returns to the Unit Operations editor.

B. USAGE

The software library "libopord.a" should be built and installed in the directory "/common/lib." You will also need the header file "libopord.h" which should be installed in the directory "/common/include/libinc/." If these files are not installed, then you need to do a "gmake" in the LibOpord source directory. See the ModSAF Software Architecture Design and Overview Document for additional information [LORAa93].

C. FUNCTIONS

The following are public functions and their description, along with the meaning of the arguments, and the meaning of the return values (if any).

5. opord_init

```
extern void opord_init()
```

This function is currently a no-op. It was intended as an initialization routine for the libopord library, to allocate memory to the various data structures, but this is done within the following function, opord_create_editor.

6. opord_create_editor

```
extern OPORD_EDITOR_PTR
     opord_create_editor ( data_path, reader_flags,
                           gui, tactmap, tcc,
                           map_erase_gc, sensitive,
                           refresh_event, db, select,
                           exit_fcn, exit_arg )

     char                  *data_path;
     int32                  reader_flags;
     SGUI_PTR               gui;
     TACTMAP_PTR            tactmap;
     COORD_TCC_PTR          tcc;
     GC                     map_erase_gc;
     SNSTVE_WINDOW_PTR      sensitive;
     CALLBACK_EVENT_PTR     refresh_event;
     PO_DATABASE           *db;
     SELECT_TOOL_PTR        select;
     OPORD_EXIT_FUNCTION    exit_fcn;
     ADDRESS                exit_arg;
```

data_path specifies the directory where the data files are expected.

reader_flags specifies flags to be passed to **reader_read** when reading data files.

gui specifies the SAF GUI.

pvd specifies the PVD.

tactmap specifies the tactical map.

tcc specifies the map coordinate system

map_erase_gc specifies the GC (graphics context) which can erase things from the tactical map.

sensitive specifies the sensitive window for the tactical map.

refresh_event specifies the event which fires when the map is refreshed.

db specifies the persistent object database.

select specifies the select tool.

exit_fcn specifies the operations order exit function that is called when the operations order editor is exited. This should be defined in the library that initializes this function. In this case, libunits initializes libopord, and the exit function is "units_ops_resumed", which allows the Unit Operations editor to be displayed.

exit_arg specifies the arguments to be passed with the exit function. In this case, the pointer to the libunits editor data structure is passed back to resume the unit operations order editor.

"opord_create_editor" creates the operations order editors. The data files are read either from '.' or the specified data path, depending upon the reader flags. The reader flags are the same as in **reader_read**. The return value is a pointer to the operations order editor structure, which contains the information required to generate and maintain the editors and their associated data. Additionally, this function initializes the "LibTaskUtil" library to allow for assignment of tasks without showing their associated editors.

7. opord_set_unit

```
extern void opord_set_unit( opord_gui, unit)

     OPORD_EDITOR_PTR       opord_gui;
     ObjectID              *unit;
```

opord_gui specifies the operations order editor structures.

unit specifies the persistent object identification of the unit.

This function sets the unit identification to the selected unit from the Unit Operations editor. The user has clicked the mouse on a unit on the PVD, which has brought up the Unit Operations Editor specifying this unit. This value is passed to the Operations Order editor to keep track of to which unit this operations order applies.

8. opord_state

```
extern void opord_state( opord_gui, mode, state )

      OPORD_EDITOR_PTR      opord_gui;
      SGUI_MODE_PTR         mode;
      SGUI_MODE_STATE       state;
```

This function sets the state of the operations order editor. This is equivalent in functionality to edt_state(), but is needed here because the base operations order editor is not controlled by libeditor.

INITIAL DISTRIBUTION LIST

1.	Defense Technical Information Center Cameron Station Alexandria, VA 22304-6145	2
2.	Dudley Knox Library Code 052 Naval Postgraduate School Monterey, CA 93943-5101	2
3.	Chairman Ted Lewis, Code CS/Lt Computer Science Department Naval Postgraduate School Monterey, CA 93943-5000	1
4.	Professor David R. Pratt, Code CS/Pr Computer Science Department Naval Postgraduate School Monterey, CA 93943-5000	3
5.	Professor Michael J. Zyda, Code CS/Zk Computer Science Department Naval Postgraduate School Monterey, CA 93943-5000	1
6.	Professor Robert McGhee, Code CS/Mz Computer Science Department Naval Postgraduate School Monterey, CA 93943-5000	1
7.	Dr. Andy Ceranowicz Loral ADS 50 Moulton Street Cambridge, MA 02138	1
8.	Dr. S. H. Kwak Loral ADS 50 Moulton Street Cambridge, MA 02138	1
9.	STRICOM ATTN: Mr. Stan Goodman 12350 Research Parkway Orlando, FL 32826-3276	1

10. Major Howard L. Mohn 2
Bad Aibling Station
CMR 407, Box 827
APO AE 09098